VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

BOOLOVSKÉ OPERACE S POLYGONÁLNÍMI MODELY

BOOLEAN OPERATIONS FOR POLYGONAL MESHES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. ROMAN ČIŽMARIK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL ŠPANĚL, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/21247/2017/xcizma04

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Čižmarik Roman, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Boolovské operace s polygonálními modely**
Boolean Operations for Polygonal Meshes
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte dostupné materiály na téma reprezentace polygonálních sítí a typické operace ve zpracování polygonálních sítí jako jsou decimace, vyhlazení, mesh remeshing, subdivision, apod.
2. Seznamte se s existujícími přístupy a nástroji pro výpočet booleovských operací nad polygonálními modely a porovnejte jejich vlastnosti.
3. Vyberte vhodné metody a navrhněte řešení pro realizaci booleovských operací nad polygonálními modely.
4. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace metod.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručný plakát nebo video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních tří bodů zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Španěl Michal, Ing., Ph.D.**, UPGM FIT VUT
Konzultant: Kršek Přemysl, doc. Ing., Ph.D., 3Dim Laboratory
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cieľom tejto práce je vytvoriť knižnicu booleovských operácii nad 3D polygonálnymi sieťami. Knižnica musí podporovať prácu s otvorenými modelmi, nesmie mať vyššie pamäťové nároky ako je bežné u súčasných riešení a ideálne by mala podporovať prácu s viac než dvoma modelmi súčasne. Väčšina existujúcich riešení je náchylná na aritmetické nepresnosti, alebo nedokáže pracovať s otvorenými modelmi. V riešení bola použitá metóda adaptívnych booleovských operácii, ktorá zaobchádza so vstupnými dátami ako s adaptívnymi plochami. Metóda predpokladá, že vstupné modely môžu byť lokálne upravované a žiaden konkrétny polygón nie je obzvlášť dôležitý. Namiesto počítania presných priesečníkov jednotlivých polygónov, je vstupná sieť v oblasti preniku zjemnená, kolidujúce polygóny sú odstránené a vzniknuté diery sú následne zacelené. Výhodami tohto prístupu je robustnosť voči aritmetickým nepresnostiam, podpora otvorených modelov, možnosť zvýšiť presnosť výsledku na úkor času výpočtu a schopnosť riešiť hraničné situácie ako sú koplanárne a takmer zhodné regióny. Vytvorené riešenie poskytuje tri booleovské operácie: zjednotenie, rozdiel a prienik.

Abstract

The aim of this work is to create a library for Boolean operations on 3D polygonal meshes. Resulting library has to support open models, its memory requirements shouldn't exceed those of existing solutions and it should, ideally, support multiple models. Most of the existing solutions are vulnerable to arithmetic inaccuracies, or do not support open meshes. The solution is based on Adaptive Mesh Booleans method which treats input models as adaptive surfaces. This method assumes that input models can be arbitrarily refined and no individual polygon is particularly important. Instead of computing exact polygon intersections, the input meshes are refined in intersection regions, intersecting polygons are discarded and created holes are closed. Advantages of this approach are robustness against numerical errors, support for open meshes, possibility to trade accuracy for computation time and ability to solve cases like co-planar and near-coincident regions. The resulting library offers three Boolean operations: union, difference and intersection.

Kľúčové slová

booleovské operácie, počítačová geometria, polygonálna sieť

Keywords

boolean operations, computer geometry, polygon mesh

Citácia

ČIŽMARIK, Roman. *Boolovské operace s polygonálními modely.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Španěl, Ph.D.

Boolovské operace s polygonálními modely

Prehlásenie

Prehlasujem, že túto diplomovú prácu som vypracoval samostatne pod vedením pána Ing. Michala Španěla, PhD. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Roman Čižmarik
22. mája 2018

Poďakovanie

Chcel by som poďakovať pánovi Ing. Michalovi Španělovi, PhD. za jeho nevšedne ústretový prístup, za jeho neoceniteľné rady a pripomienky, ktorými mi pomohol pri riešení tejto práce.

Obsah

Kapitola 1

Úvod

Booleovské operácie nad polygonálnymi modelmi sú jedným zo základných problémov počítačovej geometrie. Nie je obtiažne si predstaviť, čo sa myslí pod zjednotením, rozdielom alebo prienikom dvoch trojrozmerných objektov. Avšak implementácia týchto intuitívne ľahko pochopiteľných operácií nebýva vôbec triviálna.

Booleovské operácie sa využívajú hlavne v CAD systémoch, modelovacích nástrojoch, medicínskych aplikáciách a rôznych iných programoch, ktoré pracujú s 3D modelmi. Aj napriek existencii množstva dostupných riešení a ich implementácii, ostávajú booleovské operácie slabým článkom týchto aplikácií, hlavne kvôli komplexnosti tohto problému.

Tradičný prístup výpočtu booleovských operácii sa zakladá na numericky presnom výpočte priesečníkov vstupných modelov a ich rozdelení podľa krivky prieniku. Práve tento výpočet predstavuje najväčšiu výzvu a často býva pamäťovo aj časovo najnáročnejší. V niektorých prípadoch môže dôjsť k presiahnutiu presnosti reprezentácie čísel v počítači a nesprávnemu vyhodnoteniu testu, čo býva slabinou tejto metódy.

Cieľom tejto práce je vytvoriť knižnicu booleovských operácii nad 3D polygonálnymi sieťami, ktorá sa zakladá na práci členov Autodesk Research tímu. Na vstupné modely je nazerané ako na tzv. adaptívne polygonálne siete (*adaptive meshes*). To znamená, že žiaden konkrétny trojuholník nie je zvlášť dôležitý a preto je možné vstup lokálne meniť. Zjednodušene povedané, namiesto počítania numericky presných priesečníkov sa pridá dostatočne veľa nových trojuholníkov, zmaže sa toľko trojuholníkov, aby bolo možné vstupné modely od seba oddeliť a vzniknuté časti sa následne spoja dohromady. Výhodami tohto prístupu je robustnosť voči aritmetickým nepresnostiam, podpora otvorených modelov, možnosť zvýšiť presnosť výsledku na úkor času výpočtu a schopnosť riešiť hraničné situácie ako sú koplanárne a takmer zhodné regióny.

V nasledujúcej kapitole sa nachádza stručný prehľad existujúcich prístupov k riešeniu booleovských operácii. Tretia kapitola sa zameriava na typické problémy pri ich výpočte a obsahuje výsledky testovania niekoľkých dostupných riešení. Kapitola Metóda adaptívnych booleovských operácií popisuje prístup k riešeniu booleovských operácií zvolený v tejto práci. Piata kapitola prezentuje ďalšie metódy a techniky využité pri návrhu riešenia. Z poznatkov získaných v tretej a štvrtej kapitole vyplýva kapitola Návrh knižnice na výpočet booleovských operácii, ktorá stanovuje požadované vlastnosti výsledného riešenia a navrhuje spôsob ich realizácie. Siedma kapitola sa venuje detailom implementácie. Ôsma kapitola prezentuje výsledky testovania implementovanej knižnice. Záver sumarizuje dosiahnuté výsledky a načrtáva ďalšie pokračovanie tejto práce.

Kapitola 2

Súčasné metódy a nástroje na výpočet booleovských operácií

Existujúce metódy riešenia booleovských operácií môžeme klasifikovať na základe rôznych kritérií, napríklad podľa typu vstupných a výstupných dát, alebo zvoleného prístupu. Väčšina súčasných riešení pracuje s povrchovou reprezentáciou modelov založenou na NURBS krivkách [35, 40], zakrivených plochách [23] alebo polygonálnych sieťach [12, 27, 31]. Okrem týchto najpopulárnejších reprezentácií môžeme nájsť aj iné, ako napríklad Nef [4, 16], surfel-bounded [10] alebo L-Rep [42]. Podľa zvoleného prístupu by sme mohli booleovské operácie rozdeliť do troch skupín: presná a intervalová aritmetika, volumetrické metódy a tzv. *image-based* metódy [39].

2.1 Presná a intervalová aritmetika

Tento prístup počíta booleovské operácie priamo na vstupných modeloch. V zásade sa dá hovoriť o dvoch krokoch, ktoré majú kritický vplyv na efektivitu výslednej implementácie [25].

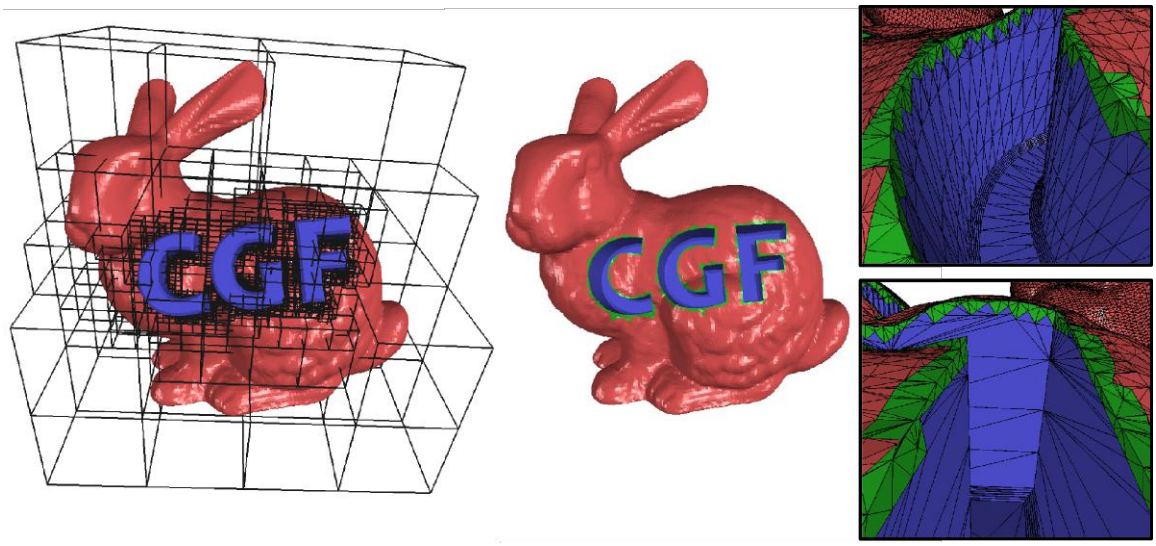
Prvým krokom je robustné a presné zistenie priesečníkov. Na čo najrýchlejšie nájdenie potenciálnych priesečníkov sa používajú rôzne akceleračné dátové štruktúry ako *Octree*, *OBB tree*, *BSP (binary space partitions)*, *bipartitné grafy* a iné [25]. Následne je však ešte potrebné medzi týmito kandidátmi presne nájsť krivku alebo cyklus priesečníku. Tento bod býva najkritickejší z hľadiska časovej aj pamäťovej náročnosti. Aby bola dosiahnutá čo najvyššia presnosť, sú používané rôzne knižnice na zvýšenie presnosti aritmetiky, ako napríklad *GNU multi precision arithmetics* [6, 31], čo ďalej zvyšuje pamäťové a časové nároky implementácie [39]. Najznámejším zástupcom tohto prístupu je knižnica CGAL. Alternatívou k exaktnému počítaniu priesečníkov je intervalová aritmetika, ktorá bola použitá napríklad v [18, 19, 15, 33].

Druhým kľúčovým krokom je korektné prevedenie požadovanej operácie nad detekovanými prelínajúcimi sa časťami vstupných modelov. Najpriamočiarejším prístupom je použitie tzv. *inside/outside* klasifikácie, ktorá jednoducho kontroluje pozície vertexov alebo plôch vzhľadom na výsledný model [25].

Tento prístup je študovaný už radu rokov (viď. [24, 28, 29]), a existuje niekoľko dostupných implementácií. Okrem už spomínanej knižnice CGAL [4], je to napríklad Cork [5], alebo Carve [3].

2.2 Volumetrické metódy

Volumetrické metódy nepočítajú booleovské operácie priamo na vstupných 3D modeloch. Vstup je najprv transformovaný do volumetrickej reprezentácie pomocou voxelizácie alebo inej techniky (napríklad adaptívny *octree*) [27] . Následne je možné relatívne jednoducho a robustne počítať booleovské operácie. Cenou je však nutnosť vzorkovania vstupných dát a strata ostrých detailov [39]. Tento problém adresuje Kobbelt et al. [22], ktorý navrhuje uložiť informáciu o normálach počas vzorkovania, čo umožňuje čiastočne zrekonštruovať pôvodné ostré hrany. Môžeme nájsť mnoho ďalších techník na zachovanie detailov pôvodných modelov, napríklad aspekt zachovania topológie [37, 38, 41], zachovanie vlastnosti manifold [30] alebo odstránenie pretínajúcich sa plôch [21]. Aj napriek tomu konverzia model – voxelizácia – model často poškodí model, alebo sa stratia jeho iné atribúty [39].

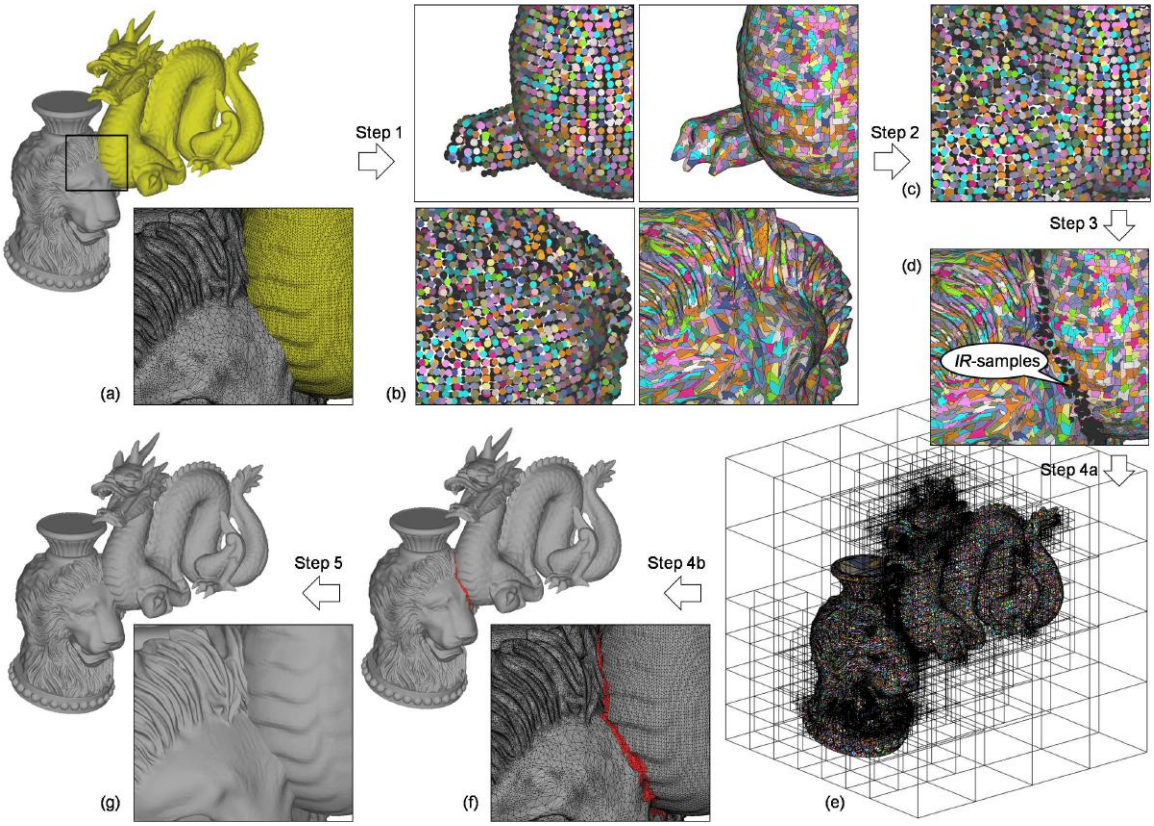


Obr. 2.1: Ukážka hybridnej metódy booleovských operácií kombinujúca volumetrickú a povrchovú reprezentáciu. Pre viac informácií viď zdroj: [27].

2.3 Image based

Táto skupina metód sa zakladá na transformácii vstupných dát pomocou *Layered Depth Images* (LDI) [34]. LDI je spôsob uloženia pohľadu na 3D scénu z jedného miesta pozorovania. Neukladá sa však iba informácia o tom, čo je z miesta pozorovania viditeľné, ale aj informácia o hĺbke v podobe 2D poľa vrstvených hĺbkových pixelov. Z každého pixelu je vrhnutý do scény lúč pozdĺž čiary priamej viditeľnosti (*line of sight*). V mieste, kde lúč pretne model je informácia o hĺbke (pripadne ďalšie potrebné informácie) uložená do 2D poľa. Takto vzniknuté vrstvy sú zoradené od najbližšej po najvzdialenejšiu plochu k miestu pozorovania, kde prvá plocha je samozrejme viditeľná pre pozorovateľa [34].

Táto oblasť je dlho študovaná a prebehol v nej rozsiahly výskum [39]. Podobne ako u predchádzajúcej skupiny je výpočet booleovskej operácie na transformovaných dátach jednoduchší, ale zdieľajú podobný problém a to závislosť na vzorkovaní a možné poškodenie vstupných dát. Wang navrhuje hybridný prístup, v ktorom sa vzorkuje iba oblasť priesečníkov [39]. Výhodou tejto skupiny metód je možnosť paralelizácie na GPU.



Obr. 2.2: Ukážka metódy približných booleovských operácií využívajúcej LDI. Pre viac informácií viď zdroj: [39].

Kapitola 3

Porovnanie existujúcich riešení na výpočet booleovských operácií

Pri výpočte booleovských operácií nad 3D modelmi sa objavuje niekoľko typických problémov. V predchádzajúcej kapitole bolo predstavených niekoľko metód, ktoré sa tieto problémy snažia adresovať a riešiť z rôznych pohľadov. Zatiaľ čo numericky presné riešenia implicitne zachovávajú detaily vstupných modelov, môžu trpieť problémom s robustnosťou. Volumetrické a *image-based* metódy tento problém nemajú, avšak sú vysoko závislé na vzorkovacej frekvencii, pričom hrozí strata detailov [25, 39, 32].

V tejto kapitole budú diskutované najčastejšie problémy pri výpočte booleovských operácií. Ďalej budú prezentované výsledky testovania štyroch vybraných dostupných implementácií. Konkrétne sa jedná o knižnice CGAL a VTK a nástroje na prácu a úpravu 3D modelov Netfabb a Meshmixer.

3.1 Typické problémy

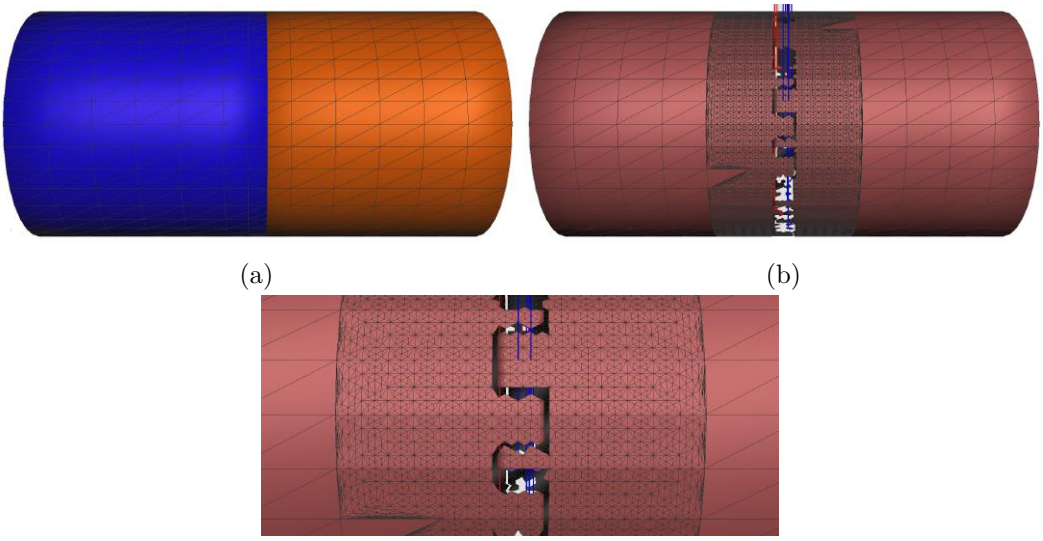
- Robustnosť

  Robustnosť je azda najdôležitejším a zároveň najproblematickejším požiadavkom na implementáciu booleovských operácií. Pod týmto pojmom rozumieme, že dané riešenie je odolné voči chybám (aritmetické, geometrické, atď.) a je schopné korektne previesť požadovanú operáciu nad hocijakým vstupom.

- Extrémne prípady

  Problematické sú najmä situácie ako koplanárne trojuholníky, alebo vertex jedného trojuholníku ležiaci v ploche iného trojuholníku, viď obrázok 3.1. V týchto prípadoch môže dôjsť pri exaktnom výpočte priesečníku k prekročeniu presnosti aritmetiky, čo vedie na nestabilné riešenie. Ak by sme uvažovali dve množiny koplanárnych trojuholníkov, tak test na existenciu priesečníku, môže pre niektoré trojuholníky skončiť pozitívne, pre iné negatívne. To typicky vedie na vznik dier alebo *non-manifold* hrán [32, 39]. Je možné použiť knižnice na zvýšenie presnosti aritmetiky, ako napríklad *GMP* [6], alebo *MIPR* (využíva knižnica Cork [5]), avšak za cenu zvýšenej pamäťovej a časovej náročnosti, pričom stále nie je zaručená stabilita výsledku.

  Podobným prípadom môžu byť aj takmer súhlasné regióny. Tento prípad si môžeme predstaviť napríklad ako dva rovnaké modely s rôznou úrovňou teselácie siete. Pre



(a) (b)

(c)

Obr. 3.1: Ilustrácia problému s aritmetickou presnosťou. Vstupom operácie zjednotenie sú dva koplanárne válce (a). Nedostatočná robustnosť voči aritmetickým nepresnostiam vedie na chybnú identifikáciu oblasti prieniku (b), v detaile na obrázku (c). Taktiež môžeme pozorovať, že pozície niektorých vertexov sa nachádzajú v nekonečne (červené a modré čiary).

implementáciu založenú na presnej aritmetike je táto situácia veľmi zložitá. Výsledkom bývajú zväčša chybné, neuzavreté modely, alebo operácia úplne zlyhá [32].

- Otvorené modely

  Otvorené modely môžu taktiež predstavovať problém pre niektoré implementácie. Uzavrený model je samozrejme ideálnym prípadom, v praxi sa však často pracuje aj s otvorenými modelmi [32].

- Zachovanie detailov

  Pre metódy, ktoré nepracujú priamo nad vstupnými dátami, ale prevádzajú ich do inej reprezentácie (vhodnejšej na výpočet booleovskej operácie), môže byť problematické zachovanie detailov modelov, prípadne iných atribútov vstupných sietí. K tomu dochádza najmä v dôsledku prevodu z a do polygonálnej reprezentácie. To je hlavným problémom volumetrických a image-based prístupov [39].

3.2 Dátová sada na testovanie

Na otestovanie vybraných existujúcich knižníc a nástrojov na výpočet booleovských operácií bola zostavená dátová sada obsahujúca sedem modelových situácií. Každá situácia obsahuje dva modely. Vizualizáciu celej dátovej sady môže čitateľ nájsť v prílohe A. Dátová sada bola zostavená na základe poznatkov prezentovaných v predchádzajúcej sekcii a v spolupráci s firmou 3DimLaboratory s.r.o., na základe ich praktických skúsenosti. Vybrané prípady sa zameriavajú na situácie, ktoré sú v praxi problematické a pri ktorých väčšina implementácií zlyháva. Nasleduje stručný popis jednotlivých testov:

1. Pretínajúce sa sféry
2. Netriviálny prienik
3. Jednoduchý a zložitý model
4. Modely s veľkým počtom trojuholníkov
5. Koplanárne polygóny
6. Otvorený a uzavrený model
7. Dva rovnaké modely s rozdielnou úrovňou teselácie, navyše obsahujúce diery

3.3 Testovanie existujúcich knižníc a nástrojov na výpočet booleovských operácií

Vybrané knižnice CGAL, VTK a nástroje Netfabb a Meshmixer boli otestované na vyššie uvedenej testovacej sade. Merané boli štyri kritériá: úspešnosť operácie, doba trvania výpočtu, maximálna spotrebovaná pamäť a korektnosť výstupného modelu. Za korektný výstup sa považuje model, ktorý odpovedá očakávanému výsledku, neobsahuje diery, ani *non-manifold* trojuholníky. Výskyt *self-intersecting* trojuhlníkov je nežiadúci. Je potrebné uviesť, že na overenie korektnosti výstupných modelov bol použitý software Meshlab. Je možné, že pri teste na *self-intersection* môže dôjsť k aritmetickým nepresnostiam a v niektorých prípadoch je výsledok testu nesprávny. Preto je výskyt niektorých *self-intersecting* trojuholníkov akceptovateľný, ak sa trojuholníky viditeľne neprelínajú. Ďalej uvádzam parametre počítača, na ktorom boli testy realizované, keďže majú vplyv na dobu trvania výpočtu: procesor Intel Core i7-5500U 2.40 GHz, 8GB RAM, grafická karta nVidia 920M, SSD harddisk. Nasleduje zhodnotenie výsledkov testovania. Podrobné výsledky testovania nájde čitateľ v prílohe C.

CGAL

Ako prvá bola testovaná knižnica CGAL. Táto knižnica je napísaná v jazyku *C++* a je dostupná aj ako *open-source*. Okrem výpočtu booleovských operácií obsahuje mnoho ďalších algoritmov na prácu s počítačovou geometriou [4]. CGAL je používaný v celej rade rôznych projektov a je považovaný za jednu z najpresnejších knižníc na výpočet booleovských operácií.

V testovaní si dokázal poradiť s väčšinou prípadov. CGAL však nedokáže previesť booleovské operácie nad otvorenými modelmi, takže testy číslo 6 a 7 boli neúspešné. Test číslo 7 bol následne modifikovaný tak, že diery v modeloch boli zalepené. V tomto prípade boli operácie úspešne dokončené, ale o korektnom výsledku sa dá hovoriť iba v prípade operácie rozdiel. Prípad koplárnych plôch zvládol bez problémov s bezchybným výstupom. Najväčším problémom však bol test číslo 4, obsahujúci veľké modely (cca 1.8 milióna trojuholníkov). V tomto prípade spotreboval CGAL 6.5 GB pamäte a bol zastavený antivírusovým systémom. Po deaktivácii antivírusu došlo spustením výpočtu približne po 4 minútach k pádu operačného systému. Z výsledkov je vidieť, že s nárastom počtu trojuholníkov vstupných modelov rastie pamäťová aj časová náročnosť výpočtu. Môže však dôjsť k situácii, kedy nároky výpočtu presiahnu fyzické možnosti výpočeného stroja a operácia skončí v lepšom prípade neúspechom, v horšom pádom operačného systému.

VTK

Druhou testovanou knižnicou bolo VTK (*Visualisation ToolKit*). VTK je *open-source* kolekciou knižníc na vizualizáciu a prácu s 3D dátami, napísaná v *C++*. Podobne ako CGAL aj VTK našlo uplatnenie v mnohých projektoch [9].

V piatich testových situáciách VTK úspešne previedlo požadované operácie. Ako korektné však môžeme označiť výstupy štyroch testov, bez detekovaných *self-intersecting* trojuholníkov iba jeden (test číslo 1). V teste 2 síce operácie prebehli úspešne, výsledok však obsahoval diery a *non-manifold* trojuholníky. V testoch 5 a 7 testovací program skončil pádom. V teste číslo 4 spotrebovalo VTK iba 820MB pamäte a dokončilo požadovanú operáciu za menej ako minútu s korektným výstupom.

Netfabb

Ďalším testovaným bol komerčný software Netfabb od firmy Autodesk. Tento program sa využíva na prípravu modelov pre 3D tlač. Obsahuje radu nástrojov a funkcií na úpravu a manipuláciu s 3D modelmi, okrem iných aj booleovské operácie [7]. Nevýhodou je, že Netfabb podporuje prácu iba s uzavretými modelmi, ponúka však nástroje na opravu a zacelenie otvorených modelov. Preto tak, ako aj v prípade knižnice CGAL, boli v poslednom teste použite uzavreté modely.

Okrem testu 6, ktorý obsahuje otvorený model, dokázal Netfabb úspešne previesť všetky požadovane operácie. V teste číslo 7, pre operácie zjednotenie a prienik, boli výstupom nekorektné modely obsahujúce *non-manifold* a *self-intersecting* trojuholníky. Všetky ostatné výstupy boli korektné. Keďže sa jedná o software, nebolo možné presne určiť množstvo spotrebovanej pamäte na výpočet booleovskej operácie. K pozorovateľnému nárastu použitej pamäte aplikáciou došlo iba v teste číslo 4, kedy vzrastala využitá pamäť z bežných 800MB až na 5GB.

Meshmixer

Posledným testovaným bol voľne dostupný software Meshmixer, taktiež od firmy Autodesk. Podobne ako Netfabb aj Meshmixer je určený na prípravu a tlač 3D modelov [1]. Metóda výpočtu booleovských operácii [32], ktorou je inšpirovaná táto práca (viď kapitola 4), je využitá práve v tejto aplikácii. Meshmixer na rozdiel od Netfabb dokáže realizovať booleovské operácie aj na otvorených modeloch. Navyše má uživateľ možnosť aplikovať celú radu nastavení požadovanej booleovskej operácie [2]. Nastavenie parametrov má veľký vplyv ako na úspešnosť samotnej operácie, tak aj na kvalitu výstupného modelu. Úspešne boli realizované booleovské operácie v šiestich zo siedmich testovacích prípadov. Avšak za korektné sa dajú označiť iba výstupy v testoch 1, 4 a 6. V testoch 2, 3, 5 boli síce požadované operácie dokončené, ale výsledky obsahovali mnoho *non-manifold* hrán, alebo vôbec neodpovedali očakávanému výstupu.

Algoritmus realizácie booleovských operácií v aplikácii Meshmixer vo veľkej miere záleží na úrovni teselácie vstupných modelov [32]. Preto sú pre tento prístup problematické prípady ako test 2 a 3, kde vstupné modely majú rádovo iný počet trojuholníkov. Zvýšením úrovne teselácie v mieste prieniku (parameter *Search-Depth*) a zvýšením počtu trojuholníkov použitých na spojenie modelov (parameter *Target Edge Length*), som vo väčšine prípadov dosiahol úspešné prevedenie operácie, avšak za cenu dlhšieho výpočtu a zvýšeného počtu trojuholníkov vo výslednej sieti.

V testovej situácii číslo 3, kde sa prvý vstupný model skladá z približne 1,000 trojuholníkov a druhý z takmer 100,000, musí prebehnúť vcelku náročné predspracovanie vstupných modelov v oblasti priesečníku. Výstupný model operácie zjednotenie v tom prípade obsahoval viac než 500,000 trojuholníkov. S podobným výsledok skončil aj test číslo 2, kde vstupy mali 12 a 515 trojuholníkov, výstup až 427,000 trojuholníkov. Je však znova nutné povedať, že tento výstup silne závisí na zvolených parametroch. Pri testovaní bola vypnutá možnosť *Auto-reduce result*, ktorá má za úlohu zjednodušiť výstupnú sieť po úspešnom dokončení požadovanej operácie. Vo väčšine prípadov tento krok viedol k zničeniu správneho výsledku booleovskej operácie.

Bezproblémové boli testy číslo 1, 4 a 6, kde vstupy obsahovali podobný počet trojuholníkov. Testy 5 a 7 nedopadli úspešne. Meshmixer však ponúka zaujímavú funkciu *Handle Co-planarity*, ktorú je možne aplikovať pri operácii diferencie [2]. Táto funkcia pomáha riešiť rozdiel koplanárnych modelov. Ani za pomoci tejto funkcie sa mi však nepodarilo úspešne realizovať tieto testy. V teste 5 operácia síce prebehla, ale výsledok bol nevalidný.

Zhodnotenie

Z výsledkov testovania je zrejmé, že ani jedna zo zvolených realizácií nedokázala úspešne previesť booleovské operácie vo všetkých testových situáciách. Už na takto malej dátovej sade môžeme pozorovať problém s robustnosťou. Najlepšie obstáli knižnice CGAL a nastroj Netfabb, ktoré však nedokážu realizovať booleovské operácie na otvorených modeloch. Nasledujúca tabuľka zhrňuje výsledky testovania. Jedná sa o subjektívny názor autora vychádzajúci z jeho skúsenosti s danými knižnicami a nástrojmi počas testovania.

	CGAL	VTK	Netfabb	Meshmixer
Pamäťová náročnosť	slabé	výborné	dobré	dobré
Časová náročnosť	dobré	výborné	dobré	slabé
Robustnosť	výborné	dobré	výborné	dobré
Otvorené modely	nie	áno	nie	áno

Tabuľka 3.1: Zhrnutie výsledkov testovania.

Kapitola 4

Metóda adaptívnych booleovských operácií

V predchádzajúcich kapitolách boli predstavené a otestované rôzne metódy na výpočet booleovských operácií nad 3D objektami. Táto kapitola má za cieľ prezentovať prístup k výpočtu booleovských operácií, tzv. adaptívne booleovské operácie, ktorý navrhol R. Schmidt et al. [32].

Zakladným predpokladom tejto metódy je, že so vstupnými 3D modelmi môžeme zaobchádzať ako s adaptívnymi polygonálnymi sieťami. To znamená, že žiaden konkrétny trojuholník nie je obzvlášť dôležitý. To nám dovoľuje vstupné modely lokálne meniť. Druhou kľúčovou komponentou je tzv. „*zippering algorithm*“ – algoritmus na zacelenie dier v polygonálnej sieti. Za pomoci týchto dvoch techník sme schopní realizovať približné booleovské operácie. Na zvýšenie presnosti operácií sú následne zavedené rôzne obmedzenia.

4.1 Lokálna zmena polygonálnej siete

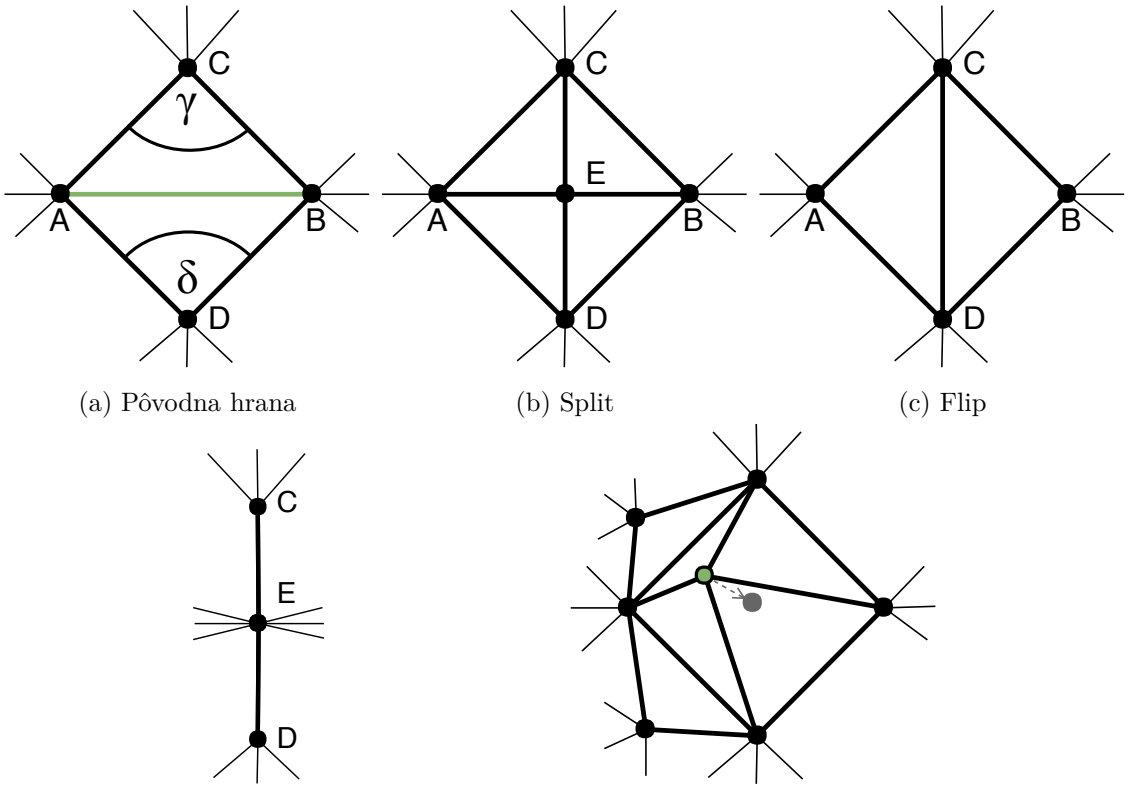
Lokálna zmena polygonálnej siete (*remeshing*, alebo *mesh refinement*) sa využíva na zníženie alebo zvýšenie hustoty siete, ale taktiež na zlepšenie kvality polygónov alebo vyhladenie povrchu modelu. Realizuje sa pomocou operácií *split*, *flip*, *collapse* a *smooth* [17, 32, 11], ilustrované na obrázku 4.1.

K samotným operáciám je však potrebné definovať sadu kontrol a obmedzení (viď. podkapitola 4.1). Ich cieľom je zamedziť strate dôležitých informácií a predchádzať poškodeniu siete vznikom *self-intersecting* alebo *non-manifold* polygónov.

Split

Operácia *split* sa využíva na rozdelenie príliš dlhých hrán. Uvažujme maximálnu dĺžku hrany l_{max}. Každá hrana dlhšia ako l_{max}, je rozdelená pridaním nového vertexu, napríklad do stredu pôvodnej hrany. Následne sú pôvodná hrana a priľahlé plochy zrušené a pridajú sa štyri nové plochy.

Táto operácia je najmenej problémová, keďže nemôže zmeniť topológiu siete ani nemôže vytvoriť *self-intersecting* alebo *non-manifold* trojuholníky.



(a) Pôvodna hrana (b) Split (c) Flip

(d) Collapse (e) Smooth

Obr. 4.1: Operácie na lokálnu zmenu polygonálnej siete.

Flip

Operácia *flip* vytvorí nové prepojenie v sieti. Vybraná hrana sa zmaže a vytvorí sa prepojenie medzi protiľahlými bodmi. Využíva sa najmä na normalizáciu valencie vertexu. Využitím techniky *flip-to-shroter*, ktorá povolí operáciu *flip* iba v prípade, že novo vzniknutá hrana bude kratšia ako pôvodná, dosiahneme ešte uniformnejšiu a kvalitnejšiu sieť. Táto operácia môže viesť k vytvoreniu *self-intersecting* a *non-manifold* trojuholníkov a mení topológiu siete.

Collapse

Operácia *collapse* sa využíva na odstránenie trojuholníkov s príliš krátkymi hranami. Ak budeme uvažovať minimálnu dĺžku hrany l_{min}, tak hrana kratšia ako l_{min} je kandidátom na *collapse*. Ďalším kritériom na prevedenie operácie *collapse* je veľkosť uhlu u protiľahlých vrcholov (na obrázku 4.1 sú to uhly γ a δ u vrcholov C a D). Ak je tento uhol menší ako $\pi/12$ pokúsime sa na vyšetrovanej hrane previesť *collapse*. Zrušenie hrany sa prevedie korektným napojením všetkých hrán z jedného koncového vertexu hrany na druhý.

Smooth

Existuje viacero variant zjemnenia povrchu polygonálnej siete. V tejto práci je použitá varianta Laplasovského zjemnenia siete, ktorá posúva vertex k centroidu okolitých plôch [32]. Polohu nového vertexu vypočítame pomocou vzťahu 4.1.

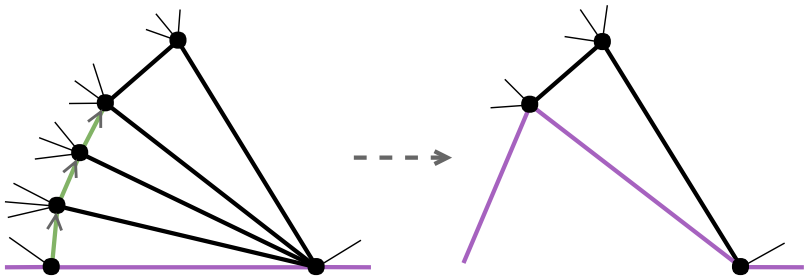
$$(1 - \alpha A)v_i + \alpha A \; c_i \tag{4.1}$$

kde A je prevrátená hodnota sumy obsahov okolitých plôch, v_i je pozícia vertexu i a α je užívateľom zadaná konštanta.

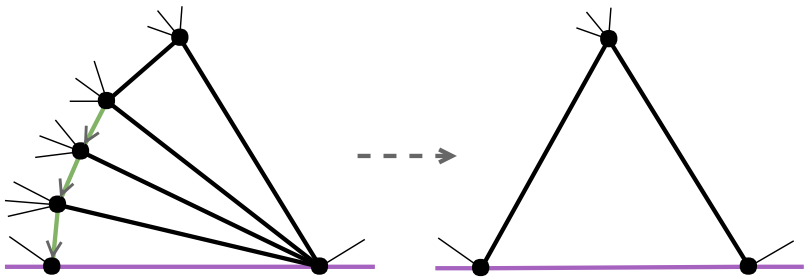
Obmedzenia

Ako bolo uvedené vyššie, vstupné modely považujeme za adaptívne siete. Platia však určité obmedzenia týkajúce sa tzv. rysových, alebo ostrých hrán (*feature edges*). Tieto hrany nejakým spôsobom reprezentujú tvar objektu a je preto žiadúce, aby boli zachované. Preto sa k týmto hranám pristupuje inak pri aplikácii operácií *flip*, *collapse* a *smooth*. Prevedú sa iba v prípade, že neporušia tvar modelu definovaný takouto hranou.

Dôležitý je aj výber vertexu, ku ktorému sa prevedie operácia *collapse* . V prípade, že je jeden z vertexov napojený na okrajovú (*boundary edge*) alebo rysovú hranu, je žiadúce, aby sa *collapse* previedol k tomuto vertexu, viď obrázok 4.2. V prípade, že sú oba vertexy napojené na okrajové hrany na tom nezáleží, keďže takouto operáciou neporušíme okraj modelu. Ak sú oba vertexy napojené na rysové hrany, operáciu nemôžeme previesť.



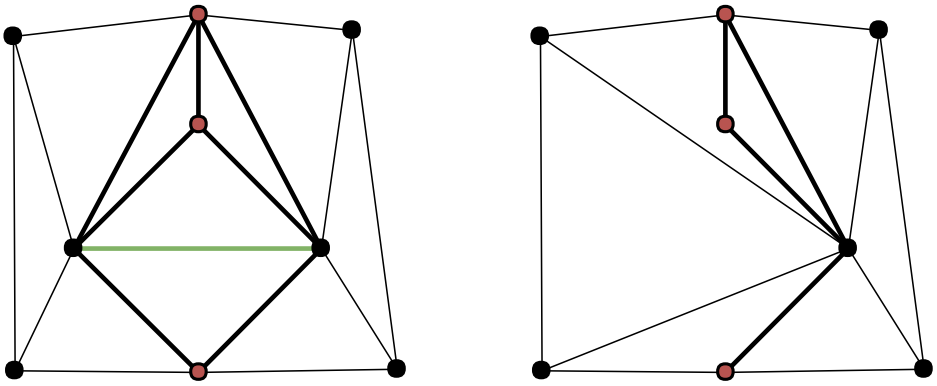
(a) Nesprávne zvolený smer operácie *collapse*.



(b) Správne zvolený smer operácie *collapse*.

Obr. 4.2: Výber smeru operácie *collapse*. Fialovou su zobrazené rysové hrany. Zelenou sú označené hrany, na ktorých chceme previesť operáciu *collapse*. V prvom prípade (a) bol nesprávne zvolený smer operácie *collapse*, čo vedie na zjavné porušenie tvaru modelu. Prípad (b) ilustruje správne zvolený smer, pričom tvar modelu ostane neporušený.

U operácie *collapse* je nevyhnutné previesť tzv. kontrolu konektivity, ktorá zamedzuje vzniku *non-manifold* trojuholníkov. Uvažujme príklad ilustrovaný na obrázku 4.3. Vyšetrovaná hrana, ktorá je kandidátom na zrušenie je označená zelenou. Jej koncové vertexy majú celkovo tri spoločné vertexy, označené červenou. Výsledkom zrušenia takejto hrany je vznik *non-manifold* trojuholníku. Kontrola konektivity spočíva v overení počtu spoločných vertexov. Pre neokrajové hrany sú to práve dva spoločné vertexy, pre okrajové práve jeden.

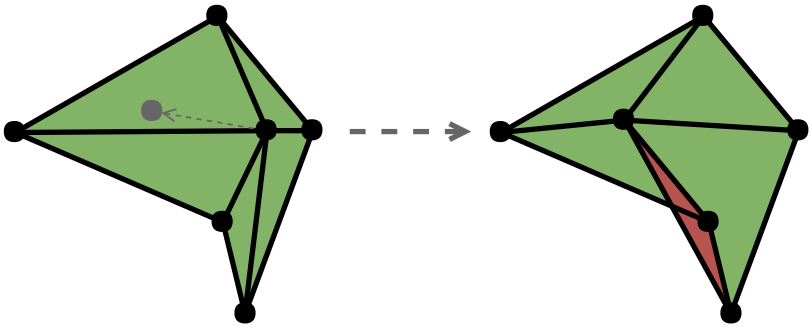


(a) Počiatočná konfigurácia.

(b) Vznik *non-manifold* trojuholníku.

Obr. 4.3: Vznik non-manifold trojuholníku prevedením operácie *collapse*. Obrázok (a) ilustruje počiatočnú konfiguráciu, kde zelenou je označená spracovávaná hrana. Spoločné vertexy koncových bodov tejto hrany sú zvýraznené červenou. Prevedenie operácie *collapse* (b) vedie na vznik *non-manifold* trojuholníku.

Po prevedení operácií *flip*, *collapse* a *smooth* sa vykoná ešte dodatočná kontrola normál priľahlých plôch. Ak sa uhol medzi normálami jednotlivých plôch zmenil o viac ako $\pi/6$, je operácia zamietnutá. Tento test zamedzuje prílišnej zmene tvaru modelu, v extrémnych prípadoch predchádza aj prevráteniu plochy, viď obrázok 4.4.



Obr. 4.4: Prevrátenie plochy spôsobené posunom bodu počas operácie *smooth*. Naľavo je zobrazená počiatočná konfigurácia, pričom normály všetkých plôch smerujú rovnakým smerom. Napravo je stav siete po prevedení operácie *smooth*. Červenou je znázornená prevrátená plocha.

4.2 Zacelenie dier v polygonálne sieti

Predpokladajme, že model $M1$ má hranicu (*boundary*) $l_1 = \{v_0, v_1, \ldots, v_n\}$ a model $M2$ má hranicu $l_2 = \{u_0, u_1, \ldots, u_m\}$. Ďalej uvažujme funkciu $nearest(l, x)$, ktorá vráti najbližší vertex z hranice l k vertexu x podľa Euklidovskej vzdialenosti. Potom môžeme zostaviť algoritmus zacelenia diery nasledovne:

while *not Bijection()* **do**
  foreach v_i *in* l_1 **do**
    Find $v_i^n \leftarrow nearest(l_2, v_i)$
    Find the new point $v_i^m \leftarrow (1-t)v_i + tv_i^n$, where $t \leq 0.5$
  end
  foreach u_j *in* l_2 **do**
    Find $u_j^n \leftarrow nearest(l_1, u_j)$
    Find the new point $u_j^m \leftarrow (1-t)u_j + tu_j^n$, where $t \leq 0.5$
  end
  Update the positions in l_1 and l_2 to the new points v_i^m and u_j^m
  Refine the meshes in the neighborhood of l_1 and l_2
end

Algoritmus 1: Zacelenie dier.

Princíp algoritmu je ilustrovaný na obrázku 4.5. Je zrejmé, že sa jedná o iteratívny algoritmus. Výpočet môžeme rozdeliť na tri logické celky:

- nájdenie najbližšieho vertexu zo susednej hranice
- aktualizácia pozície vertexov na základe nájdených najbližších susedov
- lokálna úprava siete vo vyšetrovanom hraničnom pásme.

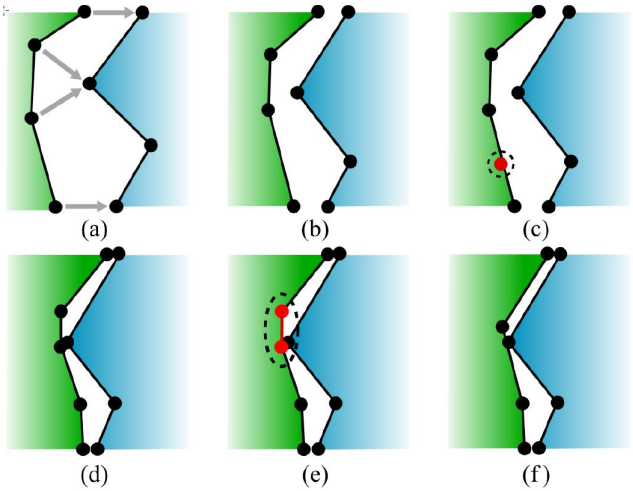
Podmienka ukončenia iterácie je nájdenie bijekcie $(l_1, nearest(l_2, v_i)) = v_i$. Nutným predpokladom na vznik bijekcie je zhodnosť veľkosti oboch hraníc, teda $m = n$. Obecne však platí, že veľkosti a tvary hraníc l_1 a l_2 sú rôzne.

Splnenie podmienky ukončenia iterácie dosiahneme aplikovaním *remeshing* operácií, z ktorých najdôležitejšie sú *split* a *collapse*. Je pravdepodobné, že viacero vertexov v_i sa bude postupne približovať k jedinému vertexu u_j, resp. naopak, čím sa budú na hranici tvoriť stále sa skracujúce hrany. V určitom momente sa však na tieto hrany aplikuje operácia *collapse* a zlúčia sa do jediného vertexu. V prípade, že sa od seba dva vertexy v rámci jednej hranice odďaľujú, pridá sa operáciou *split* nový vertex, ktorý tak zamedzí vzniku potenciálneho T-spoju (*T-junction*).

Potom môžeme vertexy v_i a $nearest(l_2, v_i)$ jednoducho nahradiť jediným vertexom a zaceliť tak dieru.

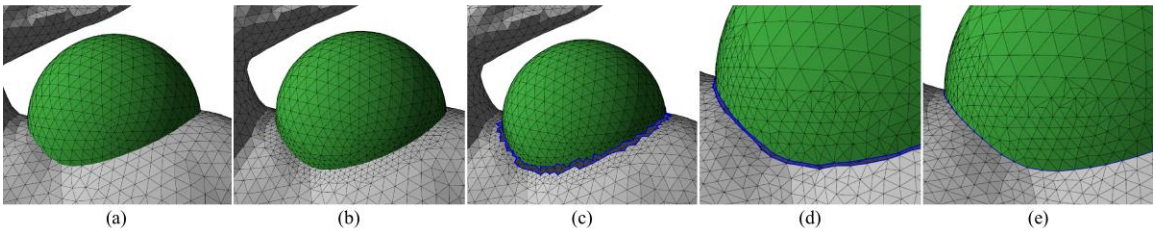
4.3 Realizácia booleovskej operácie

Ako už bolo uvedené, realizácia booleovskej operácie sa skladá z vyššie popísaných techník. Predpokladajme jednoduchý príklad, kde vstupom operácie sú dva uzavreté polygonálne modely $M1$ a $M2$, ktoré sa pretínajú iba v jednej oblasti.



Obr. 4.5: Ukážka princípu *mesh zippering* algoritmu. V prvom kroku (a) sa nájdu najbližšie vertexy z druhej hranice podľa Euklidovskej vzdialenosti. Následne (b) sa prevedie posun k nájdeným najbližším vertexom. V bode (c) môžeme vidieť aplikáciu operácie *split* na hranu, ktorá sa predĺžila. Po ďalšej iterácii (d), je jedna hrana dostatočne krátka na to, aby mohla byť aplikovaná operácia *collapse*. V bode (f) už existuje bijekcia a algoritmus môžeme ukončiť. Zdroj: [32].

Prvým krokom je nájdenie oblasti, v ktorej sa modely $M1$ a $M2$ pretínajú. Urýchlenie vyhľadávania môžeme dosiahnuť použitím akceleračnej vyhľadávacej štruktúry, napríklad *Octree*. Výsledkom je množina trojuholníkov $t1 \in M1$ a $t2 \in M2$. V nasledujúcom kroku vstupné modely v týchto oblastiach zjemníme *remeshing* operáciami, predovšetkým operáciou *split*. Tento proces sa deje iteratívne, dokiaľ nie je dosiahnutá požadovaná jemnosť detailov v oblastiach priesečníkov. Následne oblasť priesečníkov z pôvodných modelov odstránime. Vzniknú tak štyri disjunktné množiny trojuholníkov, čiže štyri otvorené modely. Následne je potrebné zmazať niektoré z novo vzniknutých modelov. To, ktoré časti zmažeme, záleží na konkrétnej booleovskej operácii. Ak by sme uvažovali operáciu zjednotenie, tak by sme odstránili tie modely, ktoré sa nachádzajú vo vnútri pôvodných modelov $M1$ a $M2$. Pre operáciu prienik by sme postupovali presne opačne. Rozdiel $M1 - M2$ by bol realizovaný odstránením časti $M1$, ktorá je vo vnútri $M2$ a časti $M2$, ktorá nie je vo vnútri $M1$. Následne je na ostávajúce novo vzniknuté modely aplikovaný algoritmus zacelenia dier, výsledkom čoho je aproximácia booleovskej operácie, ilustrované na obrázku 4.6.



Obr. 4.6: Ukážka prevedenia operácie zjednotenie na vstupe (a). V prvom kroku (b) je zjemnená sieť v oblasti priesečníku. Následne je táto oblasť odstránená (c). Vzniknutá diera je zacelená *mesh zippering* algoritmom. Výsledkom je (e). Zdroj: [32].

Uvažujme prípad, kde vstupné modely $M1$ a $M2$ majú viac ako jednu oblasť prieniku. V tomto prípade vznikne viacero nových otvorených modelov. Je preto potrebné určiť, ktoré hranice by mali byť spojené pomocou algoritmu zaceľovania dier. Riešením je jednoduchá hlasovacia schéma, kde každý vertex z každej hranice hlasuje, ktorá hranica je k nemu najbližšia. Následne sú hranice spárované po dvoch a zacelené.

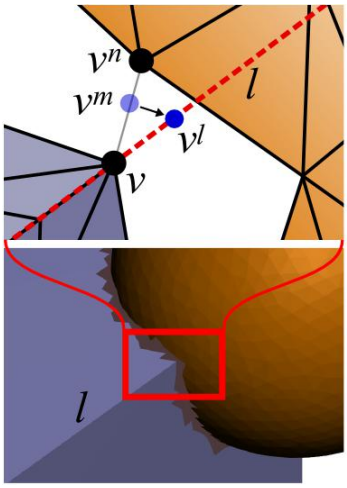
Tato metóda nevyžaduje, aby boli vstupné modely uzavreté. Jediným rozdielom oproti uzavretým modelom je určenie, či je časť vo vnútri iného modelu alebo nie. Táto prekážka sa dá odstrániť napríklad vrhnutím N lúčov z povrchu danej časti. V prípade, že aspoň $N/2$ lúčov zasiahne vnútro modelu, je táto časť klasifikovaná ako vnútorná. Ďalšou možnosťou je použitie pokročilejších metód klasifikácie, ako napríklad *Generalized Winding Number* [20].

4.4 Zvýšenie presnosti booleovskej operácie

Doposiaľ prezentovaná metóda výpočtu booleovských operácií je iba hrubou aproximáciou. Nie je ťažké si predstaviť, že aplikáciou takéhoto algoritmu by sme získali veľmi nepresné výsledky, najmä v prípadoch polygonálnych sietí s nízkym počtom trojuholníkov. Preto je nutné zaviesť určité obmedzenia, aby sme čo najviac zvýšili presnosť výslednej operácie.

Prvé obmedzenie sa týka algoritmu zaceľovania dier. Namiesto toho, aby sme v tejto fáze dovolili „voľný“ pohyb novo pridaných vertexov v 3D priestore, dovolíme im, aby sa pohybovali iba po povrchu pôvodných modelov. To znamená, že ihneď, ako je pridaný nový vertex, alebo je už existujúci vertex hranice *remeshing* operáciou posunutý, premietneme ho na povrch pôvodného modelu.

Tak, ako pri aplikácii *remeshing* operácií, aj počas zaceľovania dier je nutné zachovať pôvodné rysové hrany. Jednoduchým premietaním vertexu na povrch pôvodného modelu však môžeme takéto hrany stratiť, ako je ilustrované na obrázku 4.7. Predpokladajme, že sme vo vstupnom modely detekovali rysové hrany, ktoré tvoria úsečku l. Potom každý vertex v, pôvodne ležiaci na úsečke l, s ňou bude zviazaný. Vo fáze projekcia potom neumiestnime nový vertex v_m k najbližšiemu bodu na povrchu pôvodného modelu, ale premietneme ho na úsečku l do bodu v_l.



Obr. 4.7: Projekcia vertexu v na rysovú úsečku l. Zdroj: [32].

Takéto obmedzenia však so sebou prinášajú aj určité komplikácie. A to v prípade, že vo fáze mazania trojuholníkov, zmažeme trojuholníky priľahlé k rysovej hrane pôvodného modelu. Vtedy bude pohyb nových vertexov obmedzený na pohyb po rysovej hrane, čo môže viesť k vzniku dier vo výslednom modely, pretože zacelenie diery neprebehne správne. Môžeme tomu predísť dostatočnou úrovňou zjemnenia siete v oblasti priesečníku, alebo priamo implicitným pridaním pasu trojuholníkov okolo oblasti priesečníku. Potom nedôjde k vyššie opísanému scenáru a diera bude korektne zacelená.

Kapitola 5

Ďalšie použité metódy

Táto kapitola prezentuje ďalšie metódy a techniky využité pri návrhu knižnice. Konkrétne sa jedná o techniku *Generalized Winding Number* určenú na klasifikáciu bodov v 3D priestore z pohľadu náležitosti do vnútra alebo vonkajšku vstupného modelu. Ďalej bude popísaný test prieniku dvoch trojuholníkov v 3D priestore.

5.1 Generalized Winding Number

Metóda *Generalized Winding Number* je určená na robustnú segmentáciu objemu uzavretých ako aj otvorených polygonálnych sietí [20]. Pomocou tejto metódy dokážeme presne určiť, či sa nejaký bod nachádza vo vnútri alebo mimo vstupného modelu.

Tradičný počet závitov (*winding number*) je celé číslo so znamienkom definované pre bod P a uzavretú krivku C v 2D priestore. Udáva počet celých otočiek krivky okolo daného bodu v protismere hodinových ručičiek, viď obrázok 5.1a. Intuitívne si môžeme predstaviť, že v bode P je pozorovateľ, ktorý pozoruje bod pohybujúci sa po krivke C. Počet závitov potom udáva počet celých otočiek, ktoré pozorovateľ urobil v protismere hodinových ručičiek. Plná otočka proti smeru hodinových ručičiek pričíta jednotku, zatiaľ čo otočka po smere hodinových ručičiek odčíta jednotku. Formálne je počet závitov definovaný ako dĺžka projekcie krivky C so znamienkom na jednotkovú kružnicu okolo bodu P, vydelená 2π. Ak by sme uvažovali polárne súradnice a bod P ležiaci v počiatku, tak počet závitov vypočítame ako:

$$w(P) = \frac{1}{2\pi} \oint_C d\theta \tag{5.1}$$

Tento výpočet môžeme diskretizovať, ak prevedieme obecnú krivku C na po častiach lineárnu krivku:

$$w(P) = \frac{1}{2\pi} \sum_{i=1}^{n} \theta_i \tag{5.2}$$

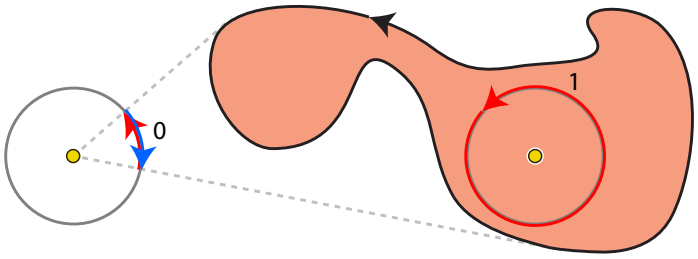
kde θ_i je uhol medzi vektormi vedenými z bodu P do dvoch za sebou nasledujúcich bodov krivky C, ilustrované na obrázku 5.1b. Generalizácia do 3D priestoru potom spočíva v zovšeobecnení krivky C na plochu S, ktorá je premietaná na jednotkovú sféru okolo bodu P. Počet závitov je potom určený vzťahom:

$$w(P) = \frac{\Omega(P)}{4\pi} \tag{5.3}$$

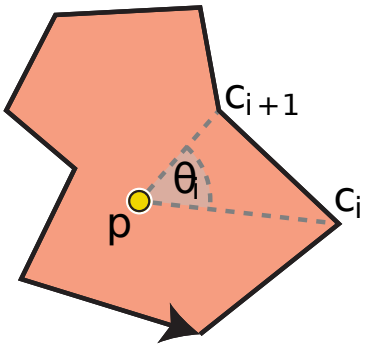
kde $\Omega(P)$ je priestorový uhol plochy S vzhľadom na bod P. Priestorový uhol predstavuje veľkosť plochy so znamienkom, ktorú vymedzuje projekcia obecnej plochy S na jednotkovú sféru so stredom v bode P. Znova môžeme previesť diskretizáciu a obecnú plochu S previesť na po častiach lineárnu plochu tvorenú trojuholníkmi a získame vzťah:

$$w(P) = \sum_{f=1}^{m} \frac{1}{4\pi} \Omega_f(P) \tag{5.4}$$

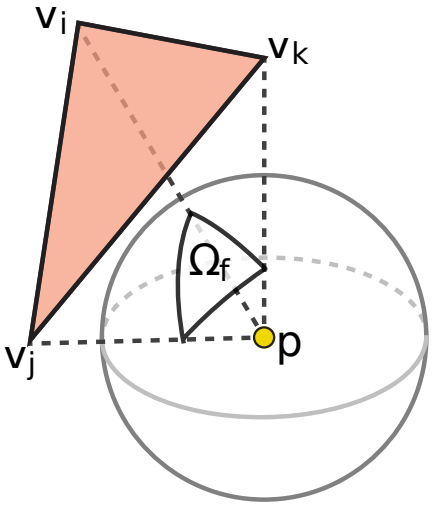
kde $\Omega_f(P)$ je priestorový uhol orientované trojuholníku, ilustrované na obrázku 5.1c.



(a) Tradičný počet závitov.



(b) Diskretizácia počtu závitov v 2D.



(c) Diskretizácia počtu závitov v 3D.

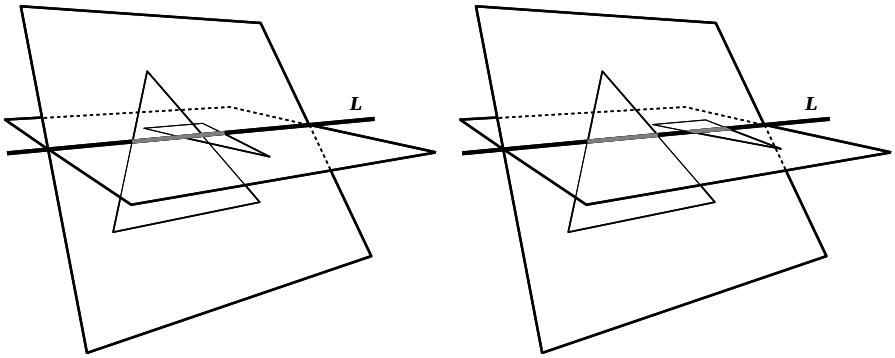
Obr. 5.1: Postupná generalizácia a diskretizácia počtu závitov do 3D priestoru využívajúc trojuholníkové plochy. Zdroj [20].

5.2 Test prieniku 3D trojuholníkov

Efektívny algoritmus testu prieniku dvoch trojuholníkov v 3D priestore je dlho skúmaný problém. Existuje niekoľko riešení ([13], [36]), avšak pri testovaní sa ukázal byť najspoľahlivejší algoritmus T. Möllera [26].

Uvažujme dva trojuholníky $T1$ a $T2$, ležiace v rovinách $\pi1$ a $\pi2$. Prienik roviny $\pi1$ a $\pi2$ je priamka L (viď obrázok 5.2). Pre všetky vrcholy trojuholníku $T1$ vypočítame vzdialenosť od roviny $\pi2$. V prípade, že je vzdialenosť nenulová a znamienko vzdialenosti sa nezmenilo, leží celý trojuholník $T1$ v jednom z pol priestorov definovaných rovinou $\pi2$. Trojuholníky $T1$ a $T2$ teda nemôžu mať priesečník a test je vyhodnotený ako negatívny. V prípade, že

vzdialenosť je pre všetky body rovná nule, sú trojuholníky koplanárne. Táto situácia sa rieši separátne. Rovnaký test sa prevedie aj pre trojuholník $T2$ a rovinu $\pi 1$. Ak test doposiaľ nebol vyhodnotený ako negatívny, je zaručené, že trojuholníky $T1$ a $T2$ pretínajú priamku L. Následne sú vypočítané intervaly $t1$ a $t2$, ktoré reprezentujú priesečníky trojuholníkov $T1$ a $T2$ s priamkou L. Ak sa tieto intervaly prekrývajú, trojuholníky $T1$ a $T2$ majú prienik.



Obr. 5.2: Test prieniku dvoch trojuholníkov. Zdroj: [26].

V prípade, že sú trojuholníky koplanárne, je potrebné previesť 2D test. Ten spočíva v kontrole priesečníku jednotlivých hrán oboch trojuholníkov. Ak sa niektoré hrany pretínajú, test je vyhodnotený ako pozitívny. V opačnom prípade je potrebné ešte otestovať, či trojuholník $T1$ neleží úplne v $T2$ a vice versa.

Kapitola 6

Návrh knižnice na výpočet booleovských operácií

Cieľom tejto práce je implementovať knižnicu obsahujúcu tri booleovské operácie nad polygonálnymi sieťami: zjednotenie, rozdiel a prienik. Knižnica bude založená na metóde popísanej v kapitole 4. V prvej časti tejto kapitoly sú definované požadované vlastnosti výstupnej knižnice. Druhá a tretia časť sa venujú návrhu spôsobu realizácie knižnice a predstavujú úpravy oproti pôvodnej metóde. Návrh realizácie bude ilustrovaný prevedením operácie zjednotenie na testovom príklade číslo 1 (viď príloha A).

6.1 Požadované vlastnosti knižnice

S ohliadnutím na nedostatky a problémy vyplývajúce z testovania existujúcich riešení a praktické požiadavky firmy 3Dim Laboratory, boli definované nasledujúce kľúčové požiadavky na vlastnosti výslednej knižnice.

- Robustnosť

  Jednou z najdôležitejších požiadaviek na výslednú implementáciu je robustnosť booleovských operácií. Výsledná knižnica by mala zvládnuť čo najširšiu škálu možných vstupných konfigurácií, od ideálneho prípadu uzavretých vstupov s rovnakou úrovňou teselácie, cez otvorené modely až po extrémne prípady ako sú koplanárne regióny. Taktiež by mala byť odolná proti chybám, ktoré sa môžu pri výpočte vyskytnúť, ako napríklad aritmetické chyby alebo vznik defektných trojuholníkov.

- Odolnosť voči aritmetickým nepresnostiam

  Výsledná implementácia by nemala byť náchylná na aritmetické nepresnosti. Tie môžu viesť k nestabilným riešeniam a nepresným alebo chybným výsledkom.

- Otvorené modely

  Je požadované, aby implementácia podporovala prácu s otvorenými modelmi. Tie sa v praxi vyskytujú pomerne často a nemôžeme sa spoliehať iba na ideálne prípady uzavretých modelov.

- Hraničné situácie

Je taktiež nutné brať zreteľ na hraničné situácie ako sú koplanárne a takmer zhodné regióny. Tieto prípady sú obzvlášť problematické a zlyhávajú v nich mnohé existujúce riešenia (viď. kapitola 3).

- Nízka pamäťová a časová náročnosť

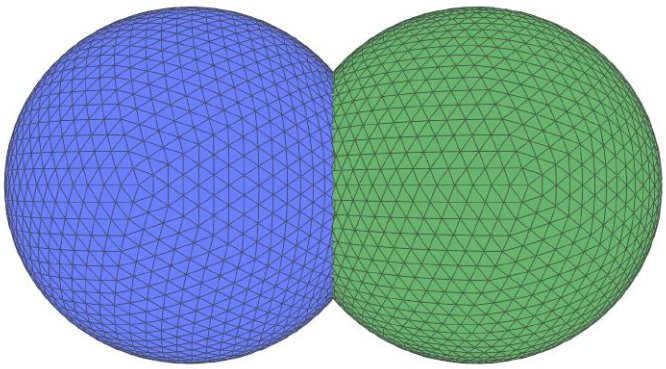
  Šetrné využívanie zdrojov je samozrejmosť. Prijateľná je možnosť zvýšenia časovej a pamäťovej náročnosti výmenou za presnejší výsledok. Výsledné riešenie by malo mať nižšie pamäťové nároky ako knižnica CGAL, ktorá je z pomedzi testovaných riešení pamäťovo najnáročnejšia. Ideálna rýchlosť výpočtu by sa mala približovať knižnici VTK.

- Podporované formáty

  Výsledná knižnica musí podporovať načítanie a ukladanie 3D modelov v bežne používaných formátoch OFF, OBJ, STL a OM.

6.2 Predspracovanie vstupných modelov

Keďže vstupné modely považujeme za adaptívne polygonálne siete, môžeme si dovoliť lokálne ich meniť a predspracovať tak, aby algoritmus zaceľovania dier pracoval čo najefektívnejšie. Úroveň predspracovania má veľký vplyv ako na rýchlosť nasledujúcej fázy, tak aj na kvalitu samotného výsledku.



Obr. 6.1: Vstupná konfigurácia testu číslo 1.

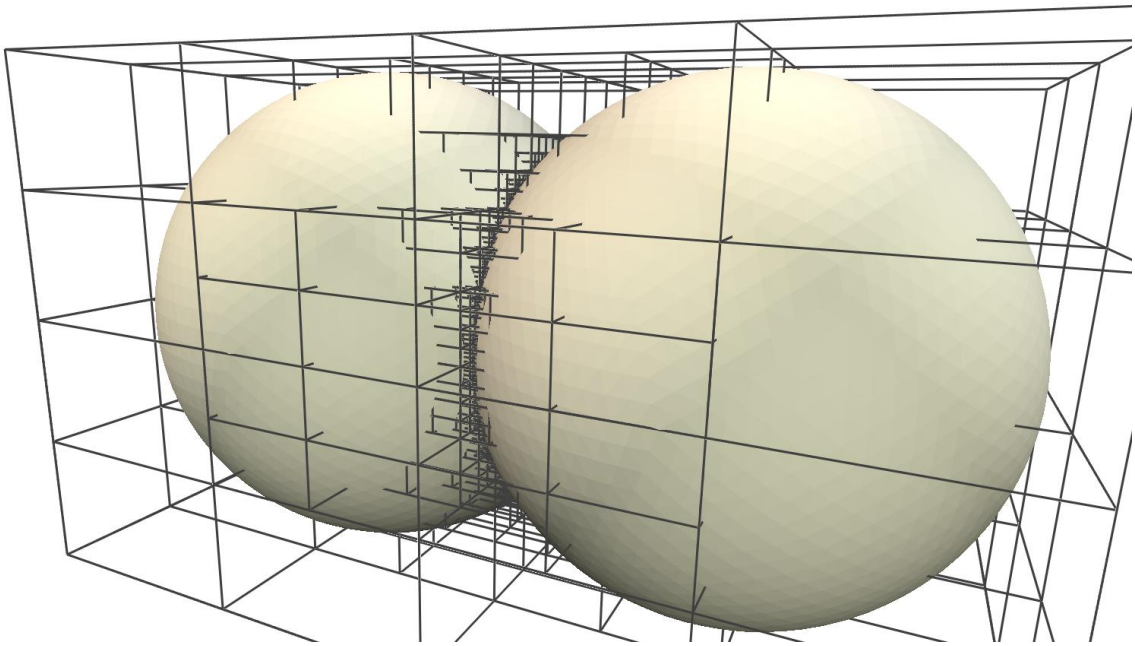
Vytvorenie výsledného modelu a akceleračnej štruktúry Octree

Zníženie časovej náročnosti dosiahneme použitím akceleračnej štruktúry *Octree*. Je to stromová dátová štruktúra, v ktorej každý uzol okrem listových, má presne 8 potomkov. *Octree* sa vytvára rekurzívnym delením 3D priestoru na oktány, pričom koreňový uzol je maximálny obaľovací kváder (*bounding box*) 3D modelu.

Octree bude navyše upravený na rýchle vyhľadávanie množiny potencionálne pretínajúcich sa trojuholníkov. Úprava spočíva v zohľadnení pôvodu jednotlivých trojuholníkov. To znamená, že v prípade, ak niektorá bunka stromu obsahuje trojuholníky, ktoré pochádzajú z rôznych vstupov, je ďalej rozdelená na osem menších buniek. Bunky sa delia, až kým nie je dosiahnutý určitý počet trojuholníkov v bunke, alebo nie je dosiahnutá maximálna hĺbka stromu. Z toho vyplýva, že delenie *Octree* bude jemnejšie v oblasti prieniku. Vďaka tomu

sme schopní veľmi rýchlo a relatívne presne identifikovať potenciálne kolidujúce trojuholníky.

Prvým krokom výpočtu booleovskej operácie je prekopírovanie všetkých vstupov do jediného modelu, ktorý bude výsledkom požadovanej operácie. Pri kopírovaní je ku každému primitívnu (trojuholník, hrana, vertex) uložený identifikátor pôvodného modelu. Následne je nad výstupnou polygonálnou sieťou vybudovaná upravená štruktúra *Octree*, viď obrázok 6.2.



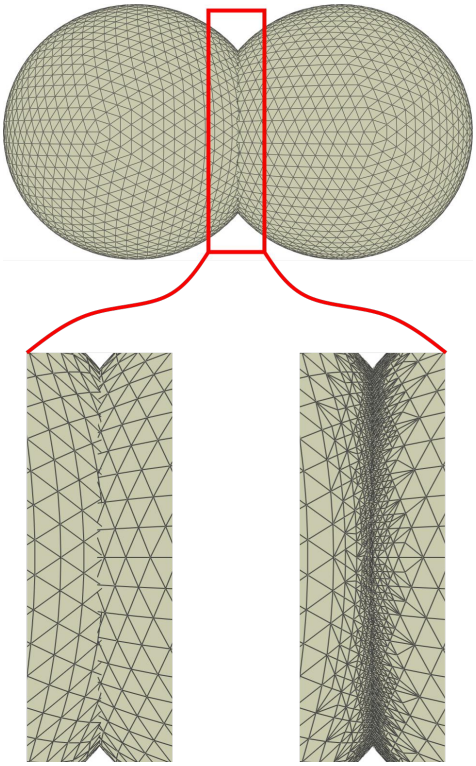
Obr. 6.2: Vybudovanie štruktúry *Octree* nad výstupným modelom. Delenie v oblasti prieniku je jemnejšie ako v zbytku modelu.

Iteratívne zjemnenie hranice

Nasledujúcim krokom je zjemnenie výstupného modelu v oblasti prieniku. Platí, že čím menšia a presnejšia je oblasť prieniku, tým rýchlejšia a presnejšia bude nasledujúca fáza zacelenia vzniknutých dier. Výstupná sieť je iteratívne zjemňovaná pomocou operácie *split*, až kým nie je dosiahnutá užívateľom požadovaná jemnosť alebo limit počtu iterácií. Zjemňovaný však nie je celý model, ale iba oblasť prieniku, pričom v každej iterácii sa aktualizuje štruktúra *Octree* použitá na vyhľadávanie priesečníkov. Vďaka tomu je detekovaná oblasť prieniku v každej iterácii presnejšia a nedochádza tak k zbytočnej teselácii v oblastiach, ktoré neobsahujú žiadne priesečníky, ilustrované na obrázku 6.3. Oproti referenčnej implementácii použitej v softwari Meshmixer vedie tento postup na nižší počet trojuholníkov vo výslednom modely, je časovo menej náročný, aj napriek nutnosti aktualizovať štruktúru *Octree*. Značný rozdiel sa prejaví najmä v situácii, kde majú vstupné modely rádovo rozdielny počet trojuholníkov, viď príklad na obrázku 6.4.

Detekcia a odstránenie prieniku

Ďalším krokom výpočtu booleovskej operácie je detekcia a odstránenie oblasti prieniku. Po dosiahnutí požadovanej úrovne teselácie je potrebné odstrániť pretínajúce sa trojuholníky.



Obr. 6.3: Zjemnenie siete v oblasti prieniku.

Tak vzniknú disjunktné časti vstupných modelov, ktoré budú v nasledujúcom kroku spojené dohromady algoritmom zaceľovania dier.

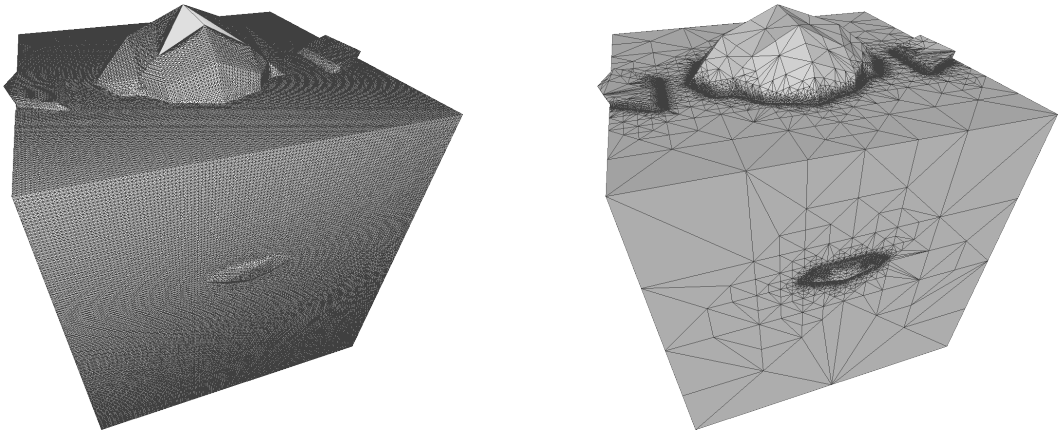
V tomto kroku však narážame na problém s aritmetickou presnosťou. Existuje niekoľko testov prieniku dvoch trojuholníkov [26], [13], [36], všetky však zdieľajú problém s presnosťou aritmetiky. Nesprávne vyhodnotenie testu priesečníku môže viesť na vznik izolovaných zhlukov trojuholníkov v oblasti prieniku, viď podkapitola 7.4. Takáto situácia potom vedie na nekorektný výstup alebo úplné zlyhanie booleovskej operácie.

Tento problém môžeme pozorovať aj u softwaru Meshmixer, ktorý mal v testovaní (test číslo 5) problém správne detekovať prienik modelov a operácia skončila neúspechom. Preto bude výsledná knižnica ponúkať tri rôzne testy detekcie oblasti prieniku, pričom každý je vhodný pre iný typ vstupnej situácie.

- Exaktný test

  Exaktný test prieniku využíva algoritmus T. Möllera popísaný v kapitole 5. Tento test je najpresnejším z implementovanej trojice testov prieniku. Je však aj najpomalší a môže mať problémy s aritmetickými nepresnosťami, najmä v prípade koplanánych trojuholníkov. Pre potreby nasledujúcej fáze (zaceľovanie dier), nie je najvyššia presnosť vždy ideálnym riešením. Je možné, že v oblasti prieniku, ktorú chceme kompletne odstrániť, môžu ostať izolované trojuholníky a výstup operácie bude nevalidný, alebo nebude odpovedať očakávanému výstupu. Naopak, exaktný test je vhodný v situácii, kedy je oblasť prienik veľmi blízko inej časti modelu. V takomto prípade by ostatné, menej presné, testy mohli nesprávne vyhodnotiť tieto časti modelov ako prienik.

- Test obaľovacích kvádrov



(a) Výstup softwaru Meshmixer. (b) Výstup navrhovanej knižnice.

Obr. 6.4: Porovnanie úrovne teselácie výstupných modelov. Oba obrázky zobrazujú výsledok operácie zjednotenie testu číslo 2 (viď kapitola 3). Výstup softwaru Meshmixer (a) obsahuje 427,746 trojuholníkov, zatiaľ čo výstup navrhovanej knižnice (b) ich obsahuje iba 26,032.

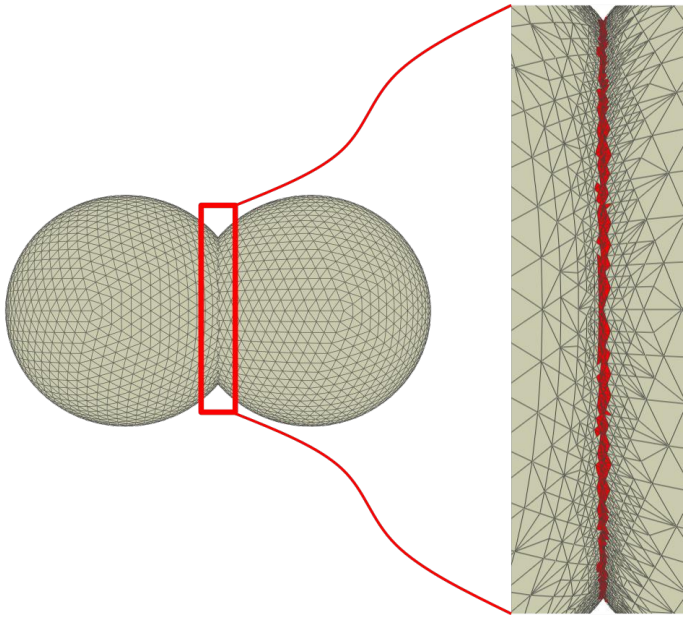
Test obaľovacích kvádrov (*bounding boxes*), ako už napovedá názov, je jednoduchým testom prieniku obaľovacích kvádrov jednotlivých trojuholníkov. Výhodou tohto testu je jeho rýchlosť. Je zrejmé, že tento test je nepresný, čo však nemusí predstavovať problém, skôr naopak, môže priniesť určité výhody. Princíp fungovania prezentovaného algoritmu booleovských operácii vyžaduje, aby odstránením oblasti prieniku vznikli dve ucelené hranice topologicky odpovedajúce kruhu. Ako už bolo spomenuté vyššie, použitím exaktného testu môžu v oblasti prieniku ostať izolované trojuholníky, čo vedie na nekorektný výstup. Použitím testu obaľovacích kvádrov sú takéto trojuholníky odstránené. Je preto možné tvrdiť, že tento test je pre väčšinu prípadov najvyhovujúcejší.

- Potenciálny prienik

  Posledný test je najrýchlejší, avšak aj najnepresnejší. Štruktúra *Octree* je využitá na rýchle vyhľadanie množiny potenciálne kolidujúcich trojuholníkov. Ak je tesel'ácia vstupných modelov prijateľne jemná a štruktúra *Octree* dostatočne hlboká, môžeme túto množinu priamo označiť za prienik, bez nutnosti ďalšieho testovania hocijakých priesečníkov. Aj keď sa jedná iba o aproximáciu prieniku, znova platí, že zavedenie určitej nepresnosti pri detekcii prieniku nemusí znamenať chybu. Tento spôsob detekcie prieniku je obzvlášť užitočný v prípade koplanárnych trojuholníkov, kde testy priesečníku obaľovacích kvádrov a exaktný test zlyhávajú.

Klasifikácia a odstránenie nepotrebných častí

V ďalšom kroku je potrebné odstrániť niektoré časti pôvodných modelov na základe požadovanej booleovskej operácie. Uvažujme množiny vstupných modelov M1 a M2. Pre operáciu zjednotenie, resp. prienik, odstránime všetky časti modelov, ktoré sú klasifikované ako „vnútorné“, resp. „vonkajšie“. Rozdiel množín M1 - M2 je realizovaný odstránením všetkých časti z M1, ktoré sú vo vnútri M2 a časti M2 klasifikovaných ako „vonkajšie“, ilustrované na obrázku 6.6.



Obr. 6.5: Detekcia oblasti prieniku pomocou testu obaľovacích kvádrov.

Klasifikácia vnútorných a vonkajších častí je realizovaná technikou *Generalized Winding Number*(viď 5.1). V prípade uzavretých vstupných modelov, je klasifikácia presná a bezproblémová. Problematická môže byť klasifikácia otvorených modelov, kde nie je možné jednoznačne určiť, ktoré časti sa nachádzajú vo vnútri a ktoré vonku. Metóda však generuje spoľahlivý a očakávaný výstup vo všetkých prípadoch. Nevýhodou je vysoká výpočetná náročnosť.

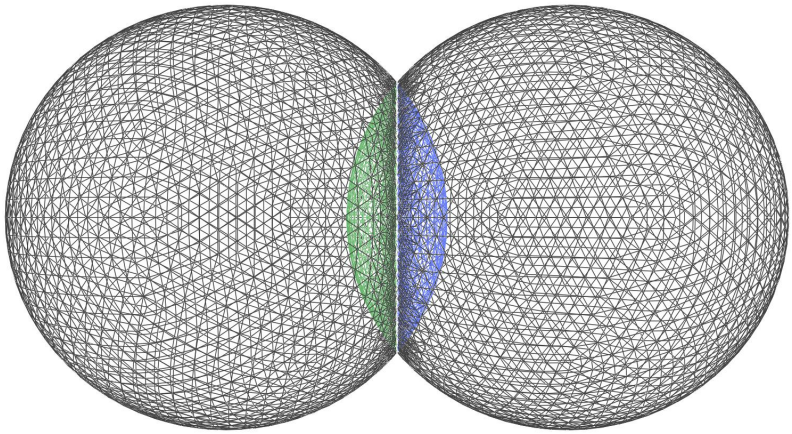
Detekcia a spárovanie hraníc

Po odstránení nadbytočných častí modelov môže vzniknúť niekoľko hraníc topologicky odpovedajúcich kruhu (*boundary loop*). Tieto hranice je nutné korektne identifikovať. Je nutné podotknúť, že nemôžeme jednoducho uvažovať všetky okrajové hrany nachádzajúce sa vo výslednom modely, keďže vstupné modely mohli byť otvorené a teda obsahovať okrajové hrany. V takom prípade by mohlo dôjsť k nezrovnalosti v počte hraníc a výstupom booleovskej operácie by mohol byť neočakávaný výstup.

V prípade, že počet novo vzniknutých hraníc je väčší ako dve, je potrebné určiť, ktoré hranice majú byť spojené dohromady pomocou algoritmu zaceľovania dier. Postačujúcou bude jednoduchá hlasovacia schéma, kde každý vertex z každej hranice hlasuje za najbližšiu hranicu z iného pôvodného modelu. Hranice, ktoré sú si navzájom najbližšie sú spárované a v nasledujúcej fáze budú spojené algoritmom zaceľovania dier.

Zjemnenie okolia hraníc a vytvorenie hraničného pásma

Posledným krokom je úprava polygonálnej siete v N-okolí detekovaných hraníc, kde N je konštanta zadaná užívateľom. N-okolie každej hranice je upravené pomocou *remeshing* operácií. Motiváciou na takúto úpravu je snaha o čo najuniformnejšiu sieť v oblasti prieniku, s približne rovnakým počtom vertexov v sparovaných dvojiciach hraníc. Za takýchto podmienok môže algoritmus zaceľovania dier pracovať najefektívnejšie a najrýchlejšie konverguje k výsledku.

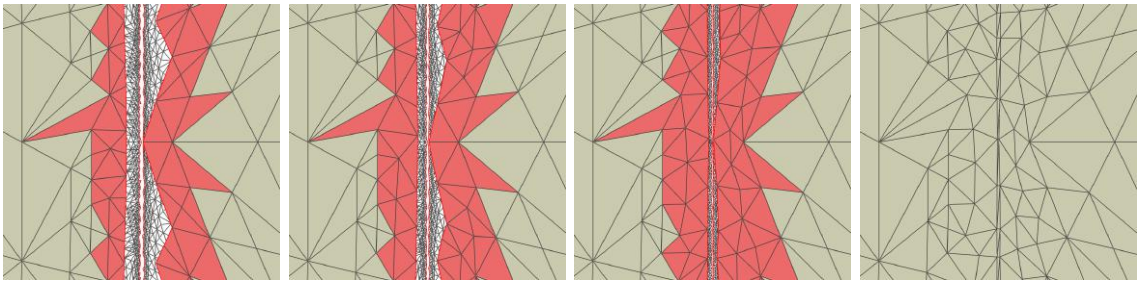


Obr. 6.6: Klasifikácia komponent. Na prevedenie operácie zjednotenie je potrebné odstrániť časti siete, ktoré sa nachádzajú vo vnútri modelu, zobrazené modrou a zelenou farbou.

N-okolie hranice je takisto považované za hraničné pásmo (*border ring*). Všetky operácie, ktoré sa budú vykonávať v nasledujúcej fáze (najmä *remeshing* operácie) sú povolené iba v tomto pásme. Vďaka tomuto obmedzeniu predídeme zbytočnej úprave polygonálnej siete v miestach, ktoré nepatria do oblasti prieniku.

6.3 Algoritmus zaceľovanie dier

Algoritmus zaceľovanie dier (*mesh zippering*) je iteratívny algoritmus, ilustrovaný na obrázku 6.7. Vstupom je dvojica hraníc, ktoré majú byť spojené dohromady. Označme tieto hranice $l_1 = \{v_0, v_1, \ldots, v_n\}$ a $l_2 = \{u_0, u_1, \ldots, u_m\}$.



Obr. 6.7: Ilustrácia algoritmu zaceľovanie dier. Hranice sa k sebe postupne približujú až kým nie je dosiahnutá bijekcia a diera nie je zacelená.

Konvergencia hraníc a reprojekcia

Prvým krokom algoritmu je nájdenie najbližšieho vertexu v susednej hranici. Pre každý vertex v_i, resp. u_i, sa vyhľadá najbližší vertex u_i^m z hranice l_2, resp. v_i^m z hranice l_1, na základe Euklidovskej vzdialenosti. Na urýchlenie vyhľadávania je znova využitá štruktúra *Octree*.

Následne sú pozície jednotlivých hraníc aktualizované na základe pozície najbližšieho vertexu v susednej hranici a konštanty α zadanej užívateľom, podľa vzťahu:

$$P_{new} = (1 - \alpha) * P_{actual} + \alpha * P_{nearest}$$

kde P_{new} je pozícia nového bodu, P_{actual} je terajšia pozícia a $P_{nearest}$ je pozícia najbližšieho vertexu v susednej hranici.

Takouto jednoduchou aktualizáciou pozície dovoľujeme vertexom „voľný“ pohyb v 3D priestore. Jednotlivé vertexy k sebe budú hľadať najkratšiu cestu a v oblasti prieniku sa teda nezachová tvar pôvodných modelov. Výsledkom je aproximácia booleovskej operácie. Na zvýšenie presnosti a kopírovanie tvaru povrchu vstupných modelov sa využíva technika reprojekcie, čiže okamžitá projekcia posunutého bodu na povrch pôvodného modelu.

Za pomoci štruktúry *Octree* nájdeme množinu trojuholníkov pôvodného modelu v okolí posúvaného bodu. Následne je potrebné nájsť najkratšiu vzdialenosť od posúvaného bodu k jednotlivým trojuholníkom. Výsledkom reprojekcie je bod s najmenšou Euklidovskou vzdialenosťou od miesta posunu.

Na zistenie najmenšej vzdialenosti medzi bodom a trojuholníkom bude využitý algoritmus D. Eberlyho [14]. V rámci návrhu boli otestované rôzne postupy na nájdenie najkratšej vzdialenosti medzi bodom a trojuholníkom v 3D priestore. Zvolený algoritmus dopadol najlepšie ako z hľadiska rýchlosti, tak aj presnosti. Autor navyše prezentuje aj variantu odolnú voči chybám zaokrúhľovania.

Je zrejmé, že zavedením takéhoto kroku sa rýchlosť konvergencie hraníc môže značne spomaliť. Užívateľ má preto možnosť krok reprojekcie vynechať v prípade, že považuje aproximáciu za postačujúci výsledok.

Ďalšie zvýšenie presnosti a zachovanie ostrých hrán v oblasti prieniku by bolo možné dosiahnuť zviazaním vertexov s rysovými hranami, viď podkapitola 4.4. Navrhovaná knižnica s touto funkcionalitou nepočíta, keďže autori označujú tento krok za problematický [32].

Úprava siete v hraničnom pásme

V nasledujúcom kroku sú hranice a ich hraničné pásma lokálne upravené pomocou *remeshing* operácií, v poradí: *split*, *collapse*, *flip* a *smooth*. Poradie aplikovania operácií môže výrazne ovplyvniť efektivitu algoritmu [32].

V tejto situácii je nutné rozlišovať tri typy hrán, s ktorými pracujeme, a síce: hraničné, rysové a ostatné. Hraničné hrany definujú otvorené okraje časti modelov, ktoré sa snažíme zlúčiť. Na tieto hrany sú aplikované iba operácie *split* a *collase* s najvyššou prioritou. To znamená, že sa neprevádzajú žiadne dodatočné kontroly vhodnosti prevedenia operácie (viď podkapitola 4.1). Takýto postup je nutný, pretože dodatočné kontroly by mohli viesť k zastaveniu posunu hranice a algoritmus by sa dostal do nekonečnej slučky. Rysové hrany, ktoré definujú tvar modelu, je potrebné spracovávať opatrne. Na tieto hrany nie je možné aplikovať operácie *flip* alebo *smooth*, pretože by viedli k porušeniu tvaru pôvodného modelu. Na všetky ostatné hrany sú aplikované všetky operácie s plnou kontrolou, aby nedošlo ku vzniku *self-intersecting* alebo *non-manifold* trojuholníkov.

Ukončenie algoritmu

Podmienkou ukončenia algoritmu zaceľovania dier je nájdenie bijekcie (viď. podkapitola 4.2). Vertexy tvoriace bijekciu sú nahradené jediným vertexom, čím sa diera zacelí. Booleovská operácia môže skončiť aj neúspechom, v prípade, že pri zaceľovaní dier je presiahnutý

limit počtu iterácií. Táto podmienka predchádza uviaznutiu algoritmu v nekonečnej slučke, v prípade výskytu neočakávanej situácie, ako napríklad zastavenie konvergencie hraníc.

Po úspešnom ukončení zaceľovania dier sa ponúka možnosť dodatočnej úpravy siete pomocou *remeshing* operácii (tzv. *post-processing*). Výsledkom by mala byť kvalitnejšia sieť s nižším počtom trojuholníkov. Testovanie však preukázalo, že tento krok vždy viedol na poškodenie inak správneho výstupu. Je to zapríčinené najmä povahou operácií *collapse* a *flip*, ktoré môžu zmeniť topológiu siete. Na rozdiel od softwaru Meshmixer navrhovaná knižnica nebude tento krok podporovať. Namiesto toho sa bude snažiť čo najšetrnejšie upraviť vstupné modely tak, aby nedochádzalo k zbytočnému nárastu počtu trojuholníkov.

Kapitola 7

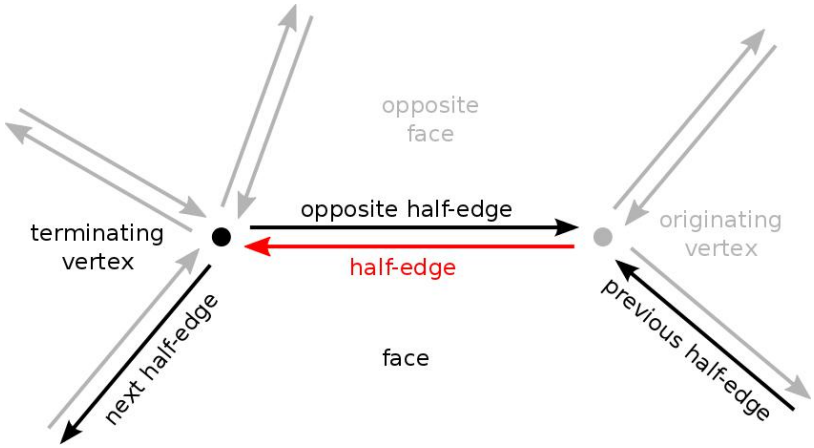
Implementácia

V tejto kapitole budú popísané detaily a špecifiká implementácie. Taktiež budú definované dodatočné kontroly konzistencie siete, ktoré odstraňujú problematické situácie zabraňujúce úspešnému ukončeniu algoritmu, s ktorými návrh nepočítal.

Zvoleným implementačným jazykom je *C++*. Hlavnými dôvodmi pre voľbu tohto jazyka je rýchlosť vykonávania a správa pamäte. Taktiež je to preferovaný implementačný jazyk vo firme 3Dim Laboratory.

7.1 OpenMesh

Na načítanie, uloženie a reprezentáciu polygonálnych sieti bude použitá *open-source* knižnica OpenMesh. Medzi kľúčové vlastnosti tejto knižnice patrí flexibilita použitia a efektívne využívanie pamäte. Dátové štruktúry implementované v tejto knižnici sú vysoko prispôsobiteľné. Je možné definovať vlastné dátové typy koordinačného systému. Polygonálnej sieti môžem pridávať a meniť atribúty, definovať dátové typy použité na ich reprezentáciu (napr. uloženie farby za pomoci `char` alebo `float`) a dynamicky za behu programu pridávať rôzne ďalšie vlastnosti polygonálnej siete.



Obr. 7.1: Štruktúra *Halfedge*. Zdroj: http://www.pointclouds.org/blog/nvcs/martin/index.php.

OpenMesh implementuje dátovú štruktúru *Halfedge* na uloženie topológie siete. Táto štruktúra uchováva informácie o topológii v hranách. Každá hrana je rozdelená na dve

Halfedge hrany (obrázok 7.1). U každej *Halfedge* hrany je jasne definovaný počiatočný a koncový vertex, priľahlá plocha, nasledujúca a protiľahlá hrana *Halfedge*. Vďaka tomu je možné jednoducho iterovať po hranách alebo vertexoch jednotlivých plôch, alebo dokonca iterovať pozdĺž otvorenej hranice. V kombinácii s uložením informácie o odstránených plochách preniku je tento spôsob iterácie s výhodou využitý pri detekcii hraníc (*boundary loop*) (viď. podkapitola 6.2). Jednoduchou sa tak stáva aj identifikácia N-okolia hranice, ktorá spočíva v N-krát zanorenej iterácii po výstupných *Halfedge* hranách jednotlivých hraničných vertexov.

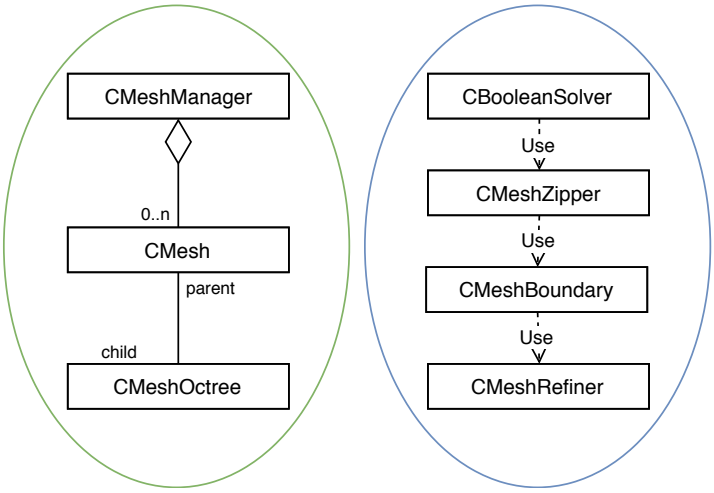
OpenMesh implementuje aj *remeshing* operácie *split*, *flip*, *collapse* a *smooth*. Tie však museli byť kompletne preimplementované. Nepostačujúce boli kontroly jednotlivých operácii, ako aj chýbajúca informácia o primitívach ovplyvnených prevedením danej operácie.

Knižnica OpenMesh dokáže pracovať so všetkými požadovanými formátmi 3D modelov. Navyše ponúka možnosť jednoduchého rozšírenia načítania a ukladania o iné formáty [8].

7.2 Štruktúra a rozhranie knižnice

Zvolený algoritmus booleovských operácií si z implementačného hľadiska nevyžaduje nijak zložitú návrhovú štruktúru. O to zložitejšia je korektná implementácia jednotlivých stavebných blokov algoritmu tak, aby nedochádzalo k porušeniu integrity a konzistencie polygonálnej siete počas výpočtu. Jedná sa predovšetkým o *remeshing* operácie ako aj dodatočné kontroly konzistencie siete. Knižnicu je možné rozdeliť na dva logické celky:

- Reprezentácia, uloženie a správa polygonálnej siete, na obrázku 7.2 zelenou.
- Realizácia booleovských operácií, na obrázku 7.2 modrou.



Obr. 7.2: Logické členenie knižnice.

Reprezentácia, uloženie a správa polygonálnej siete

Polygonálne siete zaobaľuje trieda `CMesh`. Okrem samozrejmej informácie o topológii siete si udržuje aj ďalšie dodatočné informácie potrebné pre účely realizácie booleovských operácií, ako napríklad identifikátor pôvodu u jednotlivých primitív. Aplikáciou *remeshing* operácií

však môžu primitíva vznikať a zanikať, preto je nutné dbať na korektné udržovanie ich atribútov počas celého behu algoritmu. `CMesh` ďalej uchováva indikátor náležitosti jednotlivých primitivít polygonálnej siete do hraničného pásma (*border ring*) a samotnej hranice (*boundary loop*).

Neoddeliteľnou súčasťou triedy `CMesh` je akceleračná dátová štruktúra *Octree*, implementovaná triedou `CMeshOctree`. Oproti klasickej implementácii štruktúry *Octree* je `CMeshOctree` špeciálne upravená na rýchlu identifikáciu buniek stromu, ktoré obsahujú potenciálne kolidujúce trojuholníky. Keďže výstupný model `CMesh` si u jednotlivých primitív udržuje informácie o pôvode, stačí v jednotlivých listových bunkách stromu skontrolovať počet rôznych identifikátorov pôvodu. Pri vytváraní štruktúry *Octree* sa navyše bunky označené ako potenciálne kolidujúce delia ďalej, až kým nie je dosiahnutý limitný počet trojuholníkov v jednej bunke (aktuálne 10). Je zjavné, že *Octree* bude mať väčšiu hustotu v oblasti prieniku pôvodných modelov. To je výhodné, keďže dotaz na zoznam trojuholníkov v blízkosti určitého bodu nachádzajúceho sa v tejto oblasti, vráti relatívne malý počet trojuholníkov, čo významne urýchľuje výpočet konvergencie hraníc.

Trieda `CMeshManager` má za úlohu monitorovať aktuálny stav a počet modelov `CMesh`. Ponúka rozhranie na načítanie a ukladanie modelov. `CMeshManager` je implementovaný ako *singleton*. Vďaka tomu je možné pristúpiť k aktuálne používaným modelom z hociktorého miesta v knižnici.

Realizácia booleovských operácií

Za riadenie realizácie booleovských operácii je zodpovedná trieda `CMeshBooleanSolver`. Rozhranie triedy sprístupňuje booleovské operácie `add`, `substraction` a `intersection`. Vstupom sú množiny polygonálnych sieti `CMesh` a nastavenia operácie. Z návrhu vyplýva, že realizácia booleovských operácii nebude uniformná pre každý vstup, ale bude silne závislá na zvolených parametroch operácie. Tie musí užívateľ vyplniť na základe povahy vstupných modelov. Je možné, že pri zvolení nevyhovujúcich parametrov skončí operácia neúspešne. Nasleduje zoznam implementovaných nastavení:

- Presnosť

  Presnosťou sa myslí najväčšia povolená dĺžka hrany v oblasti prieniku.

- Počet iterácií zjemňovania

  Súvisí s presnosťou. Pomocou tohto parametru je možné obmedziť úroveň zjemňovania. V prípade, že si užívateľ nie je istý presnosťou, môže využiť tento parameter (napríklad presnosť zvolí minimálnu, ale úroveň zjemňovanie obmedzí práve počtom iterácií). Parameter je nepovinný.

- Uhol definujúci rysovú hranu

  Priestorový uhol hrany meraný medzi dvoma priľahlými plochami. Pomocou tohto uhlu môžeme určiť rysové hrany, ktoré definujú tvar modelu a majú byť zachované.

- Počet iterácií algoritmu zaceľovania dier

  Tento parameter zabraňuje uviaznutiu počas zaceľovania dier v prípade, že dôjde k neočakávanej situácii a nenájde sa bijekcia. Po presiahnutí tohto limitu skončí algoritmus s chybou.

- Veľkosť hraničného pásma (*border ring*)

  Veľkosť N-okolia vzniknutej hranice, ktorá má byť použitá počas zaceľovania dier.

- Krok konvergencie hraníc

  Hodnota α použitá v kroku konvergencie hraníc (viď podkapitola 6.3).

- Test priesečníku trojuholníkov

  Užívateľ má na výber z troch testov priesečníku trojuholníkov (viď podkapitola 6.2).

- Povolenie reprojekcie

  Pomocou tohto nastavenia je možné vynechať krok reprojekcie.

K úlohám `CMeshBooleanSolver` patrí predspracovanie vstupov v podobe vytvorenia a inicializácie výstupného modelu, iteratívneho zjemnenia potenciálnej oblasti prieniku, identifikácie a odstránenia prieniku, klasifikácie a odstránenia redundantných častí, vytvorenia a spárovania hraníc. Následne sa pokúsi spárované hranice spojiť pomocou triedy `CMeshZipper`.

Trieda `CMeshZipper` implementuje algoritmus zaceľovania dier. Vstupom je dvojica hraníc, ktoré chceme zaceliť. `CMeshZipper` riadi zaceľovanie, kontroluje vznik bijekcie a vytvára konečné prepojenie, ktoré uzavrie dieru medzi hranicami. V prípade presiahnutia počtu iterácií môže predčasne ukončiť vykonávanie booleovskej operácie.

Jednotlivé hranice sú zaobalené triedou `CMeshBoundary`. Táto trieda je využitá na úpravu a manipuláciu s hranicou počas zaceľovania. K jej funkciám patrí: odstránenie izolovaných trojuholníkov, aplikácia *remeshing* operácií, posun hranice a kontrola prevrátenia trojuholníkov, reprojekcia, kontrola zloženia hranice (viď podkapitola 7.4).

Remeshing operácie implementuje trieda `CMeshRefiner`. Hlavnou úlohou tejto triedy je vykonávať operácie *split*, *flip*, *collapse*, *smooth* na základe nastavených limitov l_{min} a l_{max} a kontrolovať vhodnosť ich prevedenia. Po vykonaní požadovanej operácie je možné získať zoznam primitivít, ktoré boli počas operácie ovplyvnené. To využíva napríklad `CMeshBoundary` na aktualizáciu stavu hranice po úprave siete.

7.3 Urýchlenia a paralelizácia

Táto podkapitola bude pojednávať o prevedenej optimalizácii kódu z pohľadu rýchlosti vykonávania. Taktiež sa bude venovať možnostiam paralelizácie implementovaného algoritmu booleovských operácií.

Klasifikácia pomocou Generalized Winding Number

Na určenie vnútorných a vonkajších častí je využitá technika *Generalized Winding Number*, viď 6.2. Uvažujme vstupné modely A a B. Pre každý vertex vstupného modelu A sa vypočíta hodnota počtu závitov. Tá je následne porovnaná s limitnou hodnotou $t = 0.5$. Ak je počet závitov menší ako t, je vertex klasifikovaný ako vonkajší, inak je klasifikovaný ako vnútorný. Problém však je, že na určenie hodnoty počtu závitov pre jediný vertex modelu A je potrebné iterovať cez všetky trojuholníky modelu B a vice versa. Výpočtová náročnosť teda stúpa exponenciálne s veľkosťou vstupu. Pre väčšie modely, ako napríklad test číslo 4, je takáto klasifikácia nemysliteľná.

Preto je klasifikácia prevedená nasledujúcim spôsobom. Zo vstupných modelov sú najprv odstránené oblasti prieniku. Následne sú pomocou knižnice OpenMesh a štruktúry *Halfedge* identifikované disjunktné komponenty pôvodných modelov. Z každej komponenty je vybraných 10 náhodných trojuholníkov a ich vertexy sú klasifikované pomocou *Generalized Winding Number*. V najhoršom prípade je teda klasifikovaných 30 vertexov. Klasifikácia celej komponenty je potom vyhodnotená na základe vybraných vertexov. Bez tejto optimalizácie by implementovaný algoritmus prakticky nebol použiteľný pre väčšie modely.

Octree

Akceleračná vyhľadávacia štruktúra *Octree* nepochybne významne urýchľuje beh algoritmu. *Octree* je využité na viacerých miestach algoritmu, kde je potrebné rýchlo vyhľadať trojuholníky v určitom okolí. Príkladnými použitia sú vyhľadanie potenciálnej oblasti prieniku, alebo reprojekcia.

Spôsob, akým je štruktúra *Octree* využitá v kroku reprojekcie má kritický vplyv na rýchlosť algoritmu. V tomto kroku sa prostredníctvom *Octree* vyhľadáva množina trojuholníkov pôvodného modelu v okolí bodu, ktorý chceme premietnuť. Následne je daný bod premietnutý na každý trojuholník z tejto množiny.

Je teda zjavné, že rýchlosť výpočtu je priamo úmerná počtu nájdených okolitých trojuholníkov. Ten môžeme ovplyvniť veľkosťou okolia vyhľadávania. Ak je okolie zvolené priveľké, môže byť výsledkom relatívne početná množina a reprojekcia sa citeľne spomalí. Preto je okolie vyhľadávania volené čo najmejšie a postupne sa zväčšuje, v prípade, že je výsledná množina prázdna.

Paralelizácia

Implementovaný algoritmus je z princípu veľmi dobre paralelizovateľný. Beh algoritmu bol analyzovaný a boli identifikované určité úzke miesta (*bottleneck*), ktoré boli paralelizované pomocou knižnice OpenMP. Konkrétne sa jedná o nasledujúce časti algoritmu:

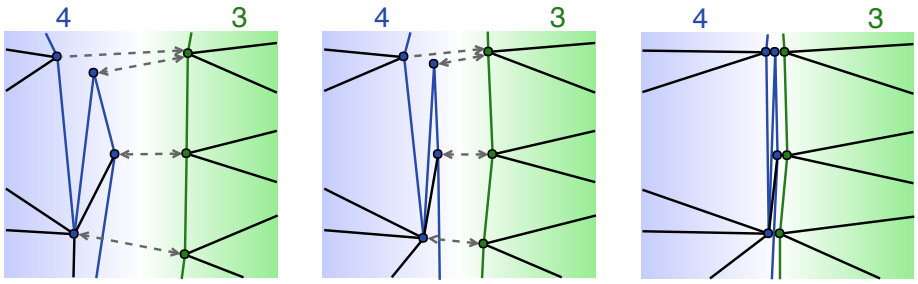
- výpočet projekcie bodu na množinu trojuholníkov
- vyhľadanie najbližšieho vertexu v susednej hranici
- vyhľadávanie v *Octree*
- krok konvergencie hraníc
- výpočet *Generalized Winding Number*

Paralelizáciou týchto častí sa beh algoritmu urýchlil o viac ako 50%. Niektoré časti však nebolo možné paralelizovať, ako napríklad krok úpravy siete v hraničnom pásme (viď podkapitola 6.3). Problematické sú *remeshing* operácie, ktoré priamo menia štruktúru dát, nad ktorými je výpočet prevádzaný. Navyše sú vykonávané v určitom poradí, *split* od najdlhšej hrany po najkratšiu, *collapse* od najkratšej hrany po najdlhšiu. Poradie vykonávania má vplyv na uniformnosť výslednej siete.

7.4 Dodatočné kontroly konzistencie siete

Počas implementácie som narazil na niekoľko problematických situácií, ktoré zabraňovali úspešnému ukončeniu operácie, alebo viedli na nekorektný výstup. Je preto nutné vykonávať dodatočné kontroly konzistencie siete, ktoré detekujú a odstraňujú potenciálne nebezpečné situácie ako sú: zloženie hranice, prevracanie trojuholníkov alebo výskyt izolovaných skupín trojuholníkov v hraničnom pásme.

Zloženie hranice



(a) (b) (c)

Obr. 7.3: Zloženie hranice: V počiatočnej situácií (a) obsahujú hranice rozdielny počet vertexov (4 a 3). Problematická je modrá hranica, ktorá obsahuje výbežok. Počas konvergencie hraníc (b) nedôjde k prevedeniu operácií *split* ani *collapse*. V určitom momente (c) ležia body na sebe, ale hranica stále obsahuje rozdielny počet vertexov, čo vedie na uviaznutie algoritmu.

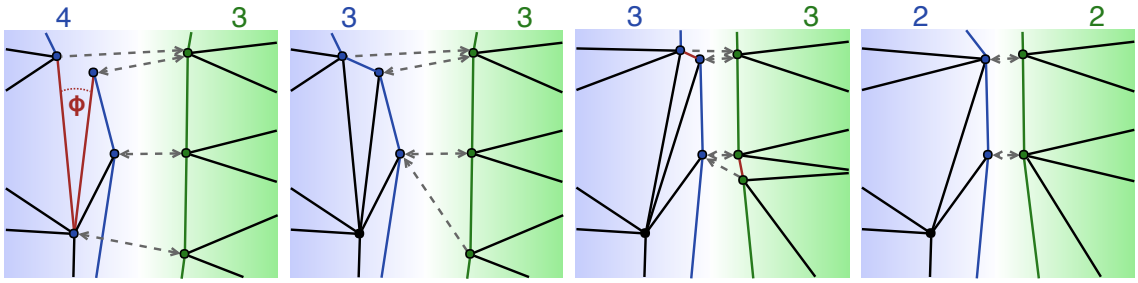
Zloženie hranice je situácia, ku ktorej môže dôjsť v kroku konvergencie hraníc a spôsobuje uviaznutie algoritmu zaceľovania dier, ilustrované na obrázku 7.3. Môže k nej dôjsť v prípade, že hranica obsahuje zakrivenia v podobe jednotlivých vyčnievajúcich trojuholníkov. V extrémnom prípade sa dva z hraničných vertexov trojuholníku približujú k rôznym bodom a môže dôjsť k preloženiu cez susednú hranu. Všetky hrany hranice pritom nie sú dostatočne krátke na prevedenie operácie *collapse* ani dostatočne dlhé na prevedenie operácie *split*. Jednotlivé body sa v určitom momente prestanú posúvať (keďže ležia priamo na sebe), ale algoritmus zaceľovania dier nemôže skončiť, pretože počet vertexov v hraniciach sa nezhoduje.

Túto situáciu je možné detekovať tak, že pre každý vertex hranice otestujeme vonkajší uhol Φ medzi priľahlými hranami hranice. V prípade, že je tento uhol menší ako určitý limit ($\pi/6$), vytvoríme medzi skúmanými bodmi nový trojuholník. Tento postup zamedzí zloženiu hranice a uviaznutiu algoritmu, ilustrované na obrázku 7.4.

Prevrátenie trojuholníkov

V prípade, že je medzi hranicami relatívne veľká medzera, môže dôjsť posunom hranice k otočeniu trojuholníku, alebo ku vzniku *self-intersections*, viď obrázok 7.5. Túto situáciu môžeme detekovať porovnaním normál trojuholníkov pred a po posune. Vertexy problematických trojuholníkov rozdelíme do dvoch kategórií:

- mobilné: vertexy, ktoré boli posunuté (patriace do hranice)

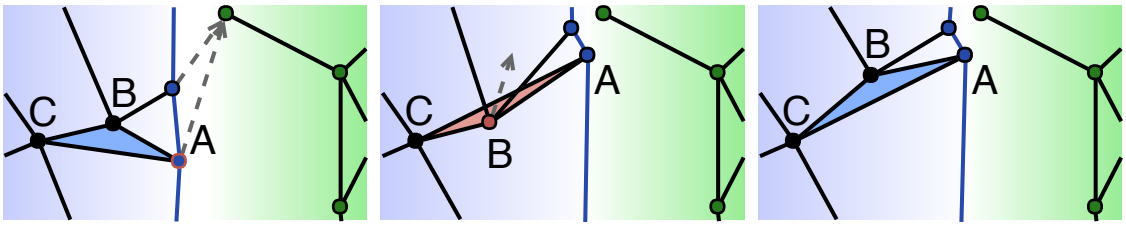


(a) (b) (c) (d)

Obr. 7.4: Riešenie zloženia hraníc: V počiatočnej situácii (a) je uhol Φ menší ako stanovený limit, preto je medzi priľahlými hranami vytvorený nový trojuholník (b). Následnou konvergenciou hraníc (c) vzniknú hrany dostatočne krátke na prevedenie operácie *collapse*. V tomto momente (d) majú hranice rovnaký počet vertexov, dochádza k bijekcii a úspešnému ukončeniu algoritmu zaceľovania dier.

- statické: vertexy, ktoré neboli posunuté (nepatria do hranice)

Následne postupne posúvame so statickými vertexami v smere pohybu mobilných vertexov, až kým nedôjdete k opätovnému otočeniu normály.



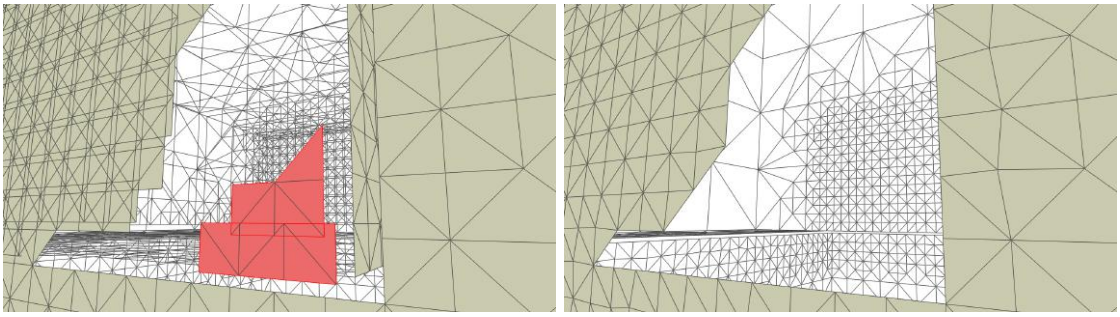
(a) (b) (c)

Obr. 7.5: Prevrátenie trojuholníku pri posune hranice. Počiatočná situácia (a) ilustruje konfiguráciu hraníc pred posunom. Povšimnime si hraničný vertex A modrej hranice. Posunom vertexu A dôjde k prevráteniu trojuholníku (b). Následným posunom statického vertexu B v smere pohybu vertexu A dôjde k opätovnému otočeniu zvýrazneného trojuholníku (c). Sieť je znova konzistentná.

Odstránenie izolovaných skupín trojuholníkov

V kroku detekcie prieniku (podkapitola 6.2) môže vplyvom aritmetických nepresností dôjsť k nekorektnému vyhodnoteniu testu priesečníkov. Následkom môže byť vznik izolovaných skupín trojuholníkov v hraničnom pásme, ktoré by však mali byť odstránené (obrázok 7.6). Takéto trojuholníky predstavujú problém pre správne fungovanie nasledujúceho kroku zaceľovania dier. Ich výskyt vedie na vznik nadbytočných hraníc (*boudnary loop*). Výsledkom booleovskej operácie je potom neočakávaný výstup, alebo zlyhanie.

Riešením tejto situácie je iteratívne odstránenie trojuholníkov v hraničnom pásme, ktoré majú viac ako jednu hranu okrajovú (*boundary*). Trojuholníky, ktoré majú viac ako jednu hranu okrajovú, môžeme charakterizovať ako nežiadúce. Je vysoko pravdepodobné, že izolovaná skupina bude obsahovať takéto trojuholníky. Navyše práve tento typ trojuholníkov



(a) (b)

Obr. 7.6: Odstránenie izolovanej skupiny trojuholníkov.

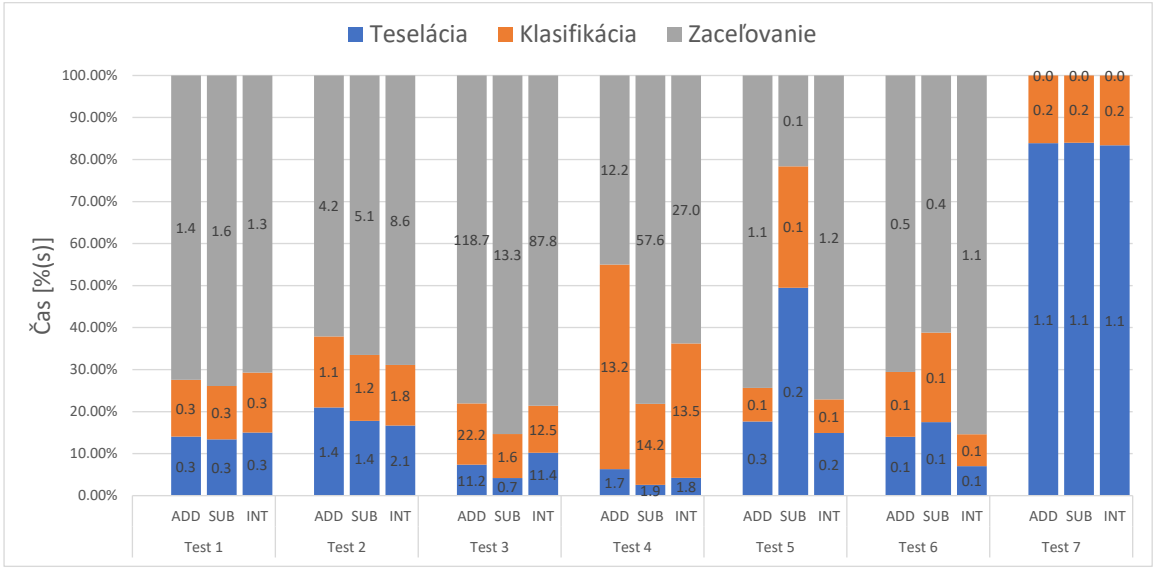
vytvára na hranici problematické výbežky popísané v predchádzajúcej sekcii. Ich odstránením dosiahneme vo väčšine prípadov elimináciu izolovaných skupín. Okrem toho vedie tento postup na uniformnejšie hranice, čo napomáha zaceľovaniu dier. Obrázok 7.6 ilustruje aplikáciu popísanej techniky.

Kapitola 8

Testovanie a experimenty

Implementovaná knižnica bola otestovaná na dátovej sade použitej pri testovaní existujúcich riešení, viď kapitola 3. Podmienky testovania boli taktiež rovnaké. Hodnotené boli samotná úspešnosť operácie, korektný výstup, pamäťová a časová náročnosť. Za korektný výstup sa pokladá model bez dier (ak to nie je očakávaný výstup) a bez *non-manifold* hrán. Model obsahujúci *self-intersecting* trojuholníky nie je hodnotený ako nesprávny, ich výskyt je však nežiadúci.

Výsledná knižnica dokázala úspešne realizovať požadované operácie vo všetkých siedmych testových prípadoch. Za korektné môžeme považovať všetky výstupy, okrem výsledkov operácií zjednotenie a prienik v teste číslo 7. Detailný komentár k jednotlivým testom nájde čitateľ v podkapitole 8.2, podrobné výsledky sú uvedené v prílohe C.



Obr. 8.1: Výsledok testovania implementovanej knižnice z pohľadu časovej náročnosti. Graf zobrazuje pomerné trvanie troch častí implementovaných booleovských operácií zjednotenie (ADD), rozdiel (SUB) a prienik (INT). U každej časti je uvedené aj jej celkové trvanie v sekundách.

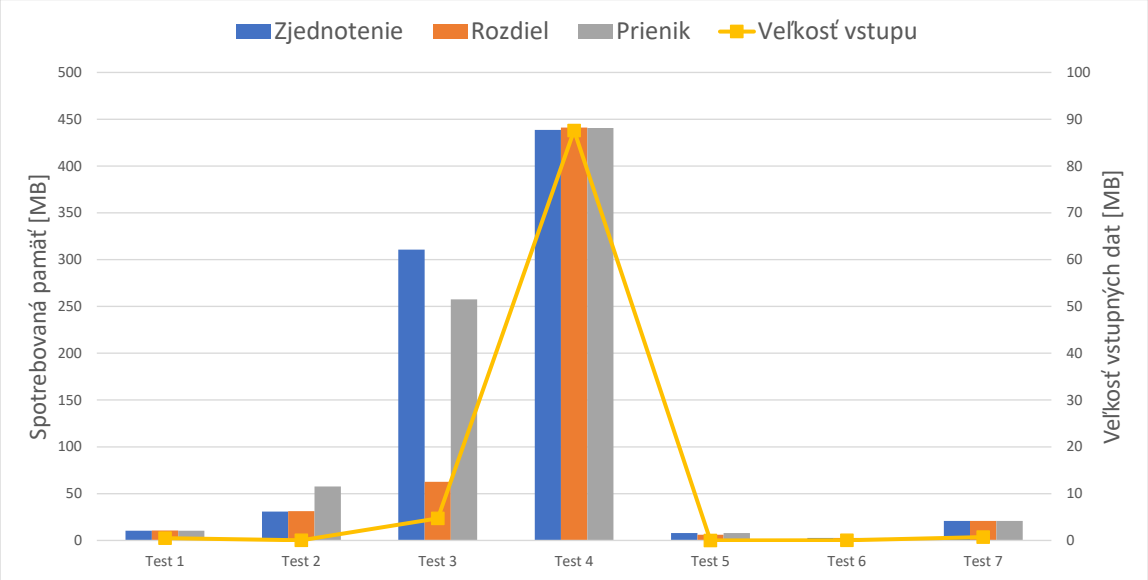
Graf 8.1 zhrňuje časovú náročnosť jednotlivých testov. Každá operácia bola rozdelená na tri logické celky, ktoré boli merané separátne, a síce:

- zjemnenie modelov v oblasti prieniku (modrá)
- klasifikácia komponent a odstránenie oblasti prieniku (oranžová)
- zaceľovanie dier (sivá)

V stĺpcoch grafu je uvedený percentuálny pomer týchto operácií ako aj celkové namerané časy. Z grafu vyplýva, že obecne časovo najnáročnejšou časťou algoritmu je zaceľovanie dier. Z meraní ďalej vyplynulo, že najdlhšie trvajúcou operáciou v rámci zaceľovania dier je reprojekcia.

Najproblematickejším a zároveň časovo aj pamäťovo najnáročnejším sa ukázal byť test číslo 3. Tento test obsahuje v oblasti prieniku jemný detail (obrázok 8.8), ktorého zachovanie je nevyhnutné na úspešné ukončenie operácie. Preto musela byť zvolená vyššia presnosť, čo prirodzene viedlo na vyššiu úroveň teselácie. To sa odrazilo ako na prvej, tak aj na druhej fáze algoritmu. Samotná hranica, ktorá bola zaceľovaná, bola čo do počtu vertexov taktiež najväčšia z celej dátovej sady. To sa premietlo v trvaní kroku zaceľovania. Pre operáciu rozdiel však tento detail nie je kritický z pohľadu ukončenia algoritmu, preto si môžeme dovoliť presnosť znížiť, čo však znamená jeho stratu. Znížením presnosti však môžeme pozorovať veľké zníženie časovej a pamäťovej náročnosti.

Zaujímavým je aj test číslo 4, kde vidíme, že trvanie klasifikácie je pomerne značné a to z dôvodu veľkosti vstupných modelov. Následné zaceľovanie je však relatívne rýchle. Ideálny prípad predstavuje operácia rozdielu v teste číslo 5. Trvanie všetkých častí algoritmu je relatívne vyrovnané. Je tomu tak z dôvodu malej medzery medzi vzniknutými hranicami a takmer rovnakému počtu vertexov v zaceľovaných hraniciach. Špeciálnym prípadom je test číslo 7. Keďže sa jedná o takmer zhodné modely, boli celé vstupné modely vyhodnotené ako oblasť prieniku a odstránené. Z tohto dôvodu ku kroku zaceľovania dier vôbec nedošlo.



Obr. 8.2: Výsledok testovania implementovanej knižnice z pohľadu pamäťovej náročnosti vo vzťahu k veľkosti vstupných dát. Stĺpce grafu zobrazujú celkové množstvo spotrebovanej pamäte jednotlivých booleovských operácií v MB. Žltá krivka udáva celkovú veľkosť vstupných dát v MB.

Graf 8.2 zobrazuje pamäťovú náročnosť booleovských operácií vo vzťahu k veľkosti vstupných modelov. Hodnoty v stĺpcoch udávajú pamäťové nároky jednotlivých operácií, žltá krivka predstavuje veľkosť vstupov. Môžeme si všimnúť, že vo väčšine prípadov odpovedajú pamäťové nároky veľkosti vstupných dát. Výnimkou je problematický test číslo 3, kde počas operácie zjednotenie resp. prienik bolo spotrebovaných až 310MB resp. 257MB pamäte, pri celkovej veľkosti vstupných modelov iba 4.7MB. Dôvodom je potreba vysokej presnosti spojená s nárastom počtu trojuholníkov v oblasti prieniku. Operácia rozdiel, ktorá bola prevedená s nižšou presnosťou spotrebovala iba 62 MB pamäte. Jemný rozdiel v pamäťovej náročnosti v teste číslo 2 pri operácii prienik je taktiež spôsobený použitím vyššej presnosti.

Namerané hodnoty časovej a pamäťovej náročnosti implementovanej knižnice môžeme porovnať s výsledkami testu knižníc CGAL a VTK (u nástrojov Netfabb a Meshmixer nebolo možné tieto hodnoty presne zmerať). Čo sa týka rýchlosti, dosahuje výsledná knižnica uspokojivé výsledky. Približne v 30% prípadov je rýchlejšia ako CGAL a v prípade operácie zjednotenie testu číslo 4 dokonca predbieha aj VTK. Pamäťové nároky sú taktiež prijateľné. Množstvo spotrebovanej pamäte je takmer vždy nižšie ako u knižnice CGAL (s výnimkou rozdielu a zjednotenia v teste číslo 2) a v polovici prípadov je dokonca nižšia ako u knižnice VTK.

8.1 Vyhodnotenie

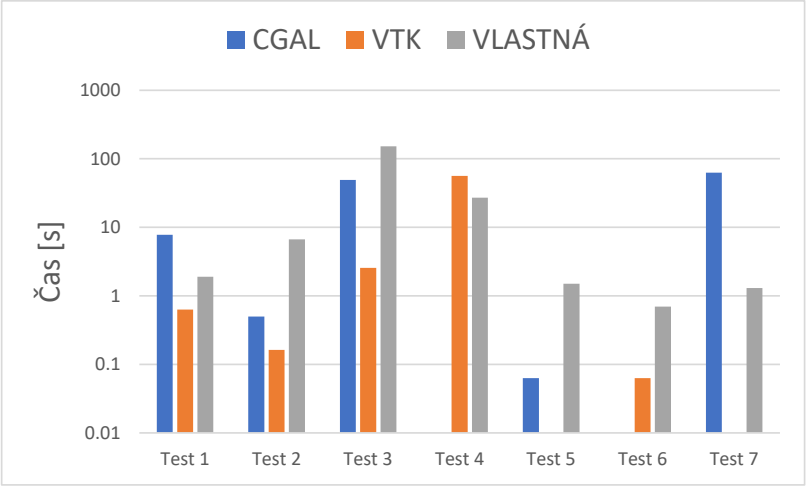
Z testovania vyplýva zaujímavá vlastnosť implementovanej knižnice. Na úkor pamäťovej a časovej náročnosti je možné zvýšiť presnosť výsledku. To môže byť výhodné v rade situácií, kedy užívateľovi nezáleží na dokonale presnom kopírovaní povrchu, alebo mu nevadí prípadná strata jemných detailov v oblasti prieniku. V takom prípade je možné docieliť ešte výraznejšieho urýchlenia vynechaním kroku reprojekcie. Je teda zjavné, že kvalita výsledku ako aj úspešnosť samotnej operácie je silne závislá na zvolených parametroch. Táto vlastnosť nemusí byť vždy vyhovujúca. V niektorých prípadoch môže byť celkom náročné správne odhadnúť ideálne parametre požadovanej operácie. V prípade využitia knižnice ako súčasť automatizovaného systému bez užívateľského vstupu, môže implementácia odhadu vyhovujúcich parametrov predstavovať prekážku.

Implementovaný algoritmus booleovských operácií je robustný voči aritmetickým nepresnostiam a dokáže sa vysporiadať s celou radou rôznych vstupných konfigurácií. Na rozdiel od väčšiny existujúcich riešení, ponúka v prípade zlyhania spôsob, akým ovplyvniť výsledok operácie. Užívateľ má vždy možnosť zmeniť použitý test detekcie prieniku, alebo zvýšiť resp. znížiť presnosť. U knižníc CGAL, VTK a softwaru Netfabb je v prípade zlyhania operácie výsledok nezvratný.

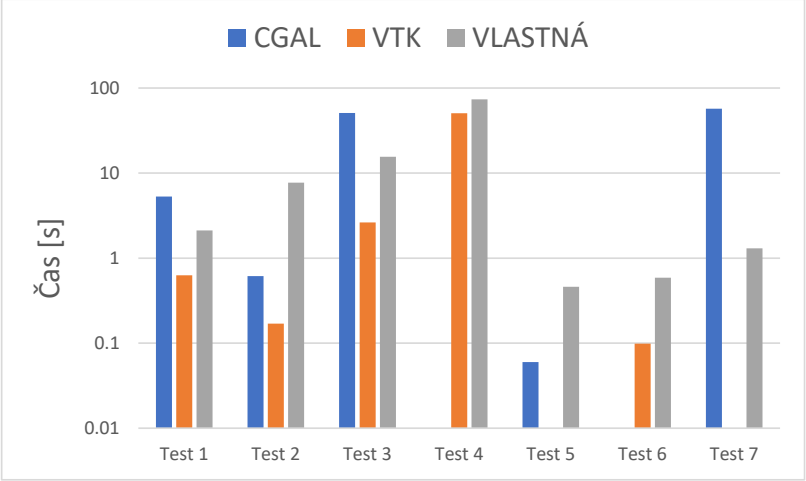
Slabou stránkou je závislosť algoritmu na úrovni teselácie. V prípade, že sa v oblasti prieniku nachádzajú dôležité detaily modelu, ktoré nemôžu byť odstránené, je nutné zvoliť dostatočne vysokú presnosť. To vedie na enormné zvýšenie počtu trojuholníkov v oblasti prieniku, čo pochopiteľne zvyšuje pamäťové a časove nároky.

Ďalšou slabinou implementovaných booleovských operácií je vznik *self-itersecting* trojuholníkov. Tento problém je spôsobený úpravou siete pomocou *remeshing* operácií počas zaceľovania dier. Pridanie ďalších kontrol môže byť predmetom ďalšieho vývoja knižnice.

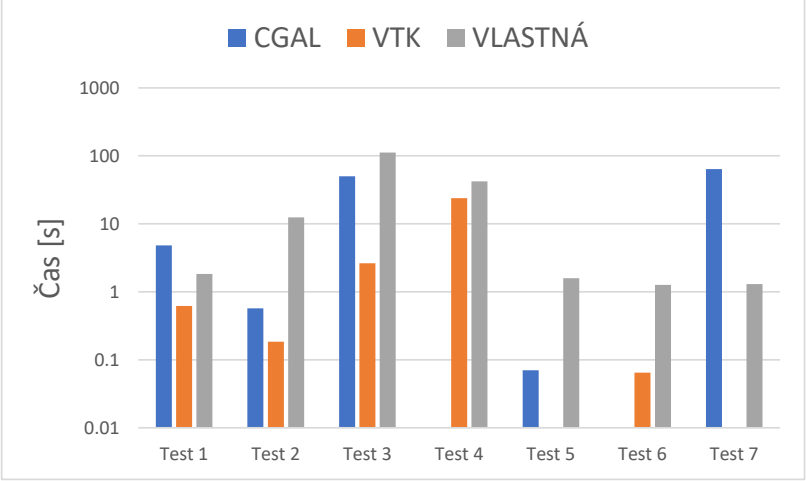
Výhodou vyplývajúcou z podstaty návrhu algoritmu je možnosť paralelizácie. Jednoduchou paralelizáciou bol beh algoritmu urýchlený o viac ako 50%. Ponúka sa teda možnosť ďalšej paralelizácie, prípadne presunu celého výpočtu algoritmu zaceľovania dier na grafickú kartu.



(a) Porovnanie operácie zjednotenie.

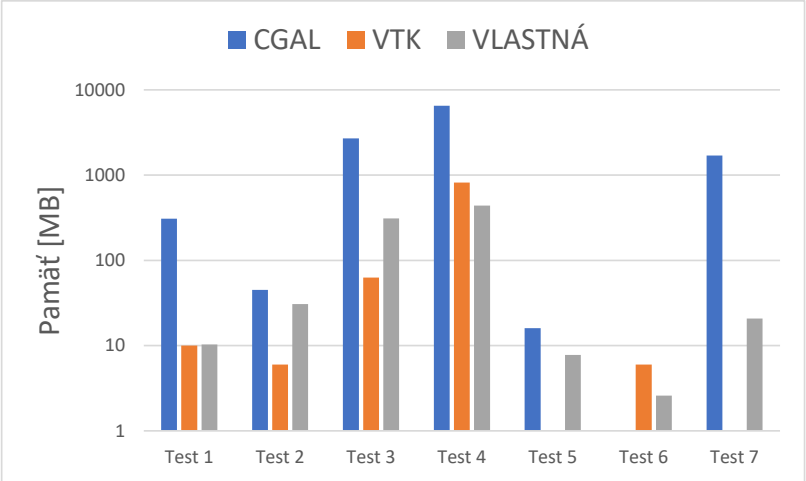


(b) Porovnanie operácie rozdiel.

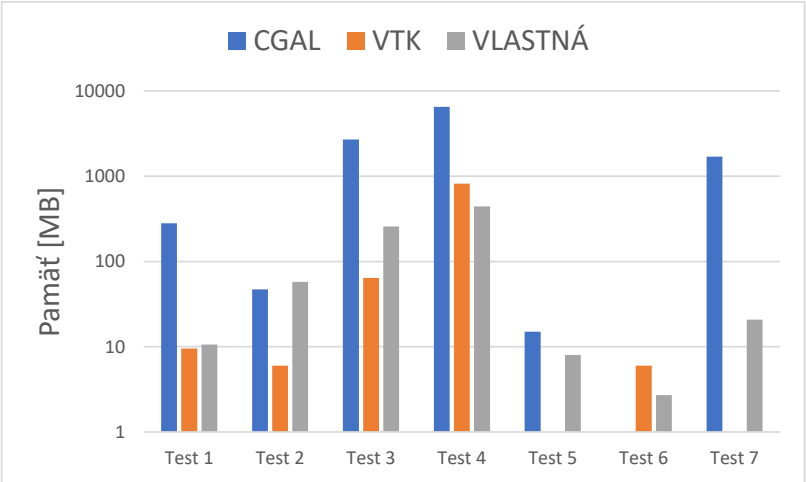


(c) Porovnanie operácie prienik.

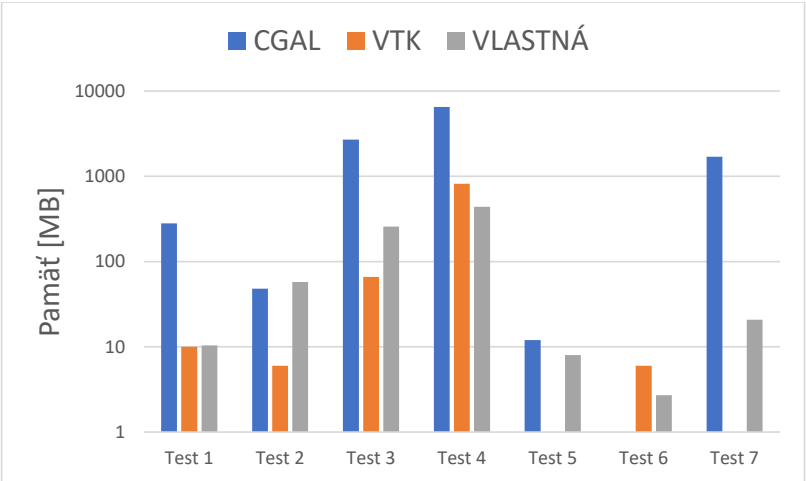
Obr. 8.3: Porovnanie časovej náročnosti jednotlivých booleovských operácií knižníc CGAL, VTK a implementovanej knižnice. Časová os je v logaritmickej mierke.



(a) Porovnanie operácie zjednotenie.



(b) Porovnanie operácie rozdiel.



(c) Porovnanie operácie prienik.

Obr. 8.4: Porovnanie pamäťovej náročnosti jednotlivých booleovských operácií knižníc CGAL, VTK a implementovanej knižnice. Časová os je v logaritmickej mierke.

8.2 Detaily testovania

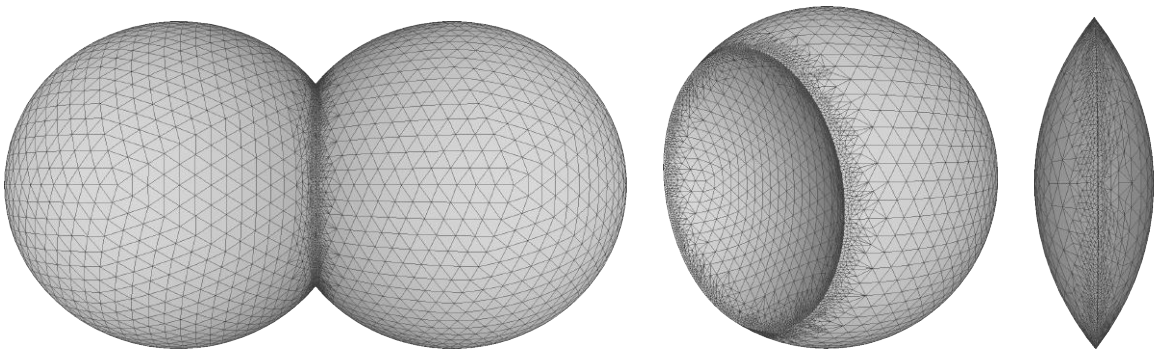
V tejto podkapitole budú detailne rozobrané a okomentované jednotlivé testované príklady. Každý test bude zhodnotený z pohľadu náročnosti pre implementovaný algoritmus. Taktiež budú popísané zvolené vstupné parametre. Ak nebude uvedené inak, predpokladá sa použitie predvolených parametrov uvedených v tabuľke 8.1.

Presnosť	0.1
Počet iterácií zjemňovania	10
Uhol definujúci rysovú hranu	45°
Počet iteráci algoritmu zaceľovania dier	40
Veľkosť hraničného pásma (*border ring*)	1
Krok konvergencie hraníc	0.2
Test priesečníku trojuholníkov	Test obaľovacích kvádrov
Povolenie reprojekcie	Áno

Tabuľka 8.1: Predvolené parametre booleovských operácií.

Test 1

Tento test bol bezproblémový pre všetky operácie. Oba vstupné modely majú rovnakú úroveň teselácie, oblasť prieniku nie je nijak komplikovaná. Jedinými použitými nastaveniami sú presnosť a obmedzenie zjemňovania, ktoré boli zvolené na základe dĺžky najdlhšej hrany vo vstupných modeloch.



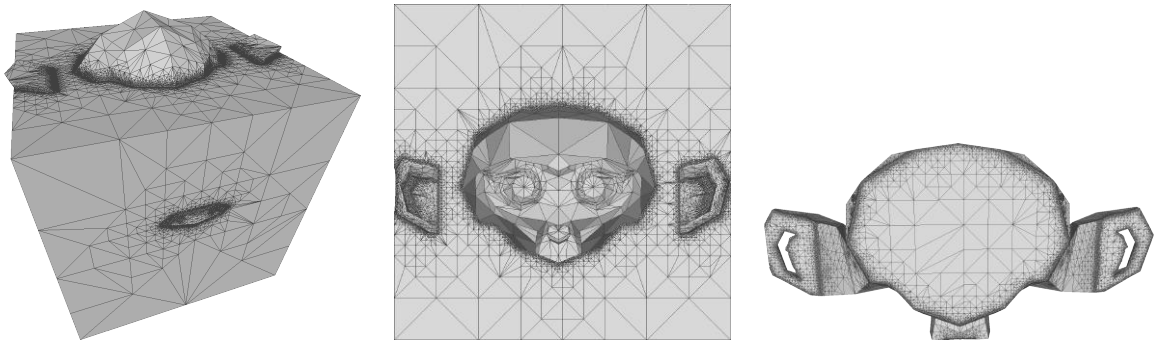
Obr. 8.5: Výsledky testu 1: zjednotenie, rozdiel, prienik (zľava doprava).

Zvolené parametre :

- Presnosť: 0.4
- Počet iterácií zjemňovania: 3

Test 2

Test 2 sa taktiež dá označiť za bezproblémový. Po odstránení oblasti prieniku vznikne celkovo 6 párov hraníc (dva vo vnútri uší opice). V prípade operácie prienik bolo experimentované s presnosťou. Pri zvýšení presnosti na 0.02 vzrástla spotreba pamäte o 11MB a



Obr. 8.6: Výsledky testu 2: zjednotenie, rozdiel, prienik (zľava doprava).

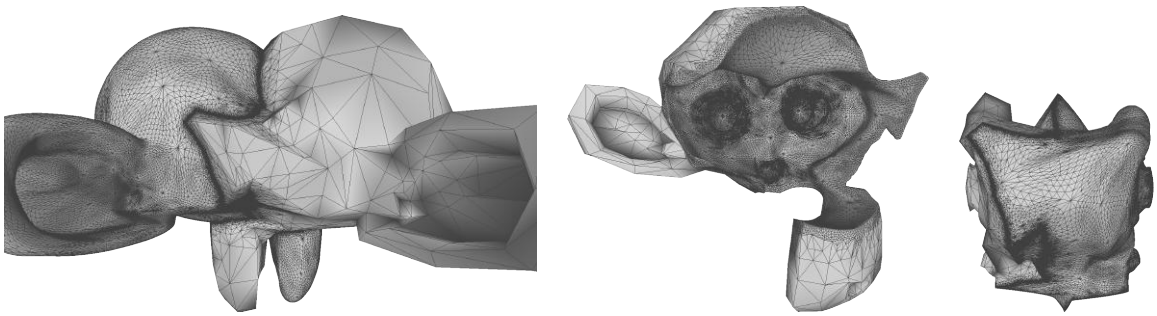
dokončenie operácie trvalo približne o 4 sekundy dlhšie. Výsledok však bol presnejší v oblasti ostrých hrán prieniku.

Zvolené parametre :

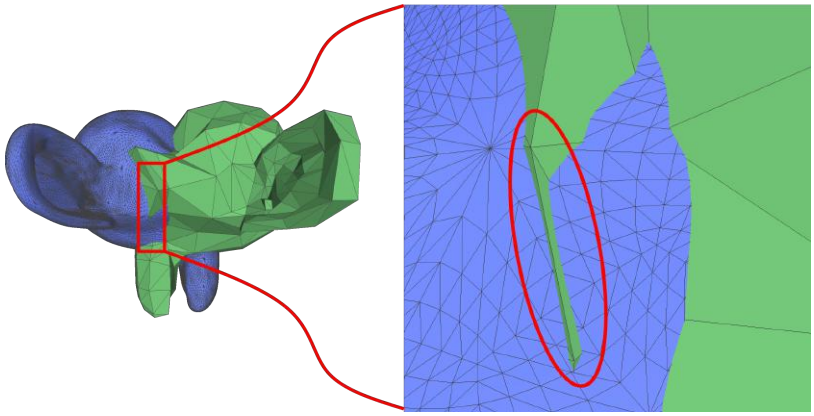
- Presnosť: 0.03
- Uhol definujúci rysovú hranu: 35°
- Test priesečníku trojuholníkov: Exaktný

Test 3

Test 3 bol najproblematickejší z celej dátovej sady. Oblasť prieniku obsahuje jemný detail (obrázok 8.8), ktorý je na úspešné dokončenie operácie potrebné zachovať. Preto bolo potrebné zvoliť relatívne vysokú presnosť, aby nedošlo k odstráneniu tohto detailu, čo sa odrazilo na pamäťovej a časovej náročnosti. Hranica je taktiež veľmi komplexná a z celej testovej sady má najväčší počet vertexov. Z dôvodu komplexnosti oblasti prieniku bolo zvolené relatívne veľké hraničné pásmo. Týmto nastavením zaručíme, že všetky trojuholníky potenciálne ovplyvnené odstránením oblasti prieniku budú zahrnuté v kroku zaceľovania dier. Operácia rozdiel bola realizovaná s nižšou presnosťou, čo znamená stratu spomínaného detailu, avšak prinieslo významné urýchlenie a nižšiu spotrebu pamäti. Samozrejme, je možné operáciu rozdiel realizovať aj s vyššou presnosťou a detail zachovať. Pre implementovaný algoritmus je tento test náročný aj z dôvodu veľmi rozdielnej úrovne teselácie vstupných modelov.



Obr. 8.7: Výsledky testu 3: zjednotenie, rozdiel, prienik (zľava doprava).



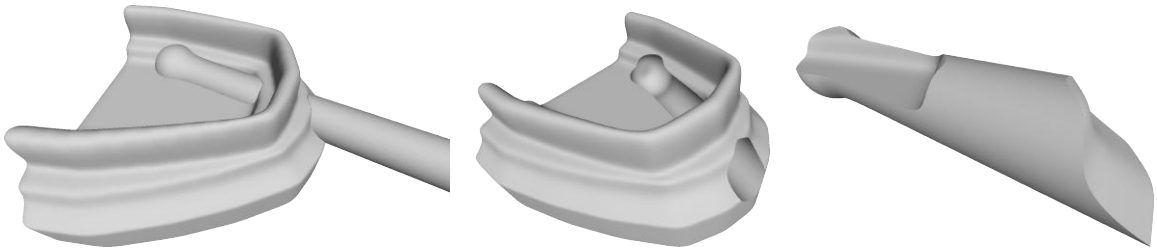
Obr. 8.8: Jemný detail v oblasti prieniku testu 3.

Zvolené parametre :

- Presnosť: 0.005
- Počet iterácií zjemňovania: 15
- Veľkosť hraničného pásma (*border ring*): 5

Test 4

Test 4 prebehol bez problémov. Vstupné modely majú podobnú úroveň teselácie. Veľkosť vstupných modelov nepredstavuje pre implementovaný algoritmus prekážku. Oblasti prieniku nie sú nijak komplikované. Dalo by sa povedať, že tento test predstavuje pre implementovaný algoritmus ideálny vstup. Časové a pamäťové nároky sú v porovnaní s ostatnými testovanými knižnicami veľmi dobré.



Obr. 8.9: Výsledky testu 4: zjednotenie, rozdiel, prienik (zľava doprava).

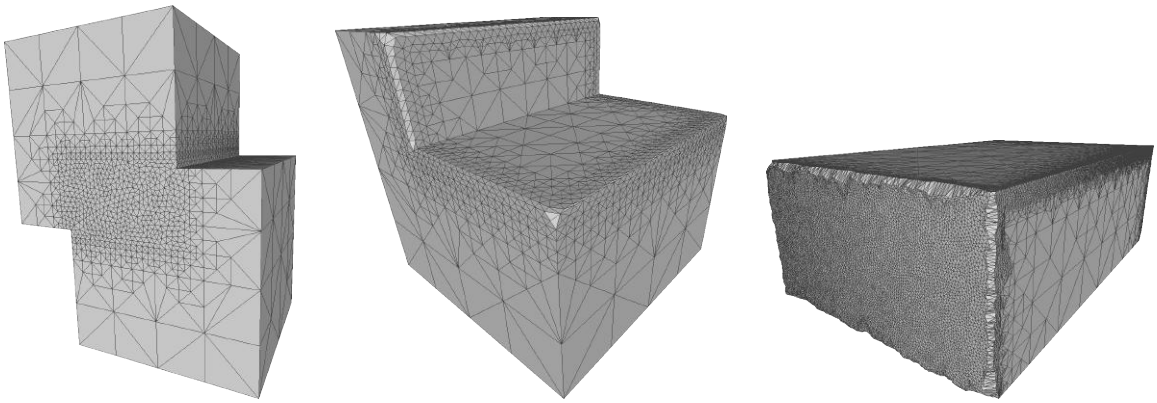
Zvolené parametre :

- Presnosť: 0.05
- Počet iterácií zjemňovania: 5
- Uhol definujúci rysovú hranu: 35°

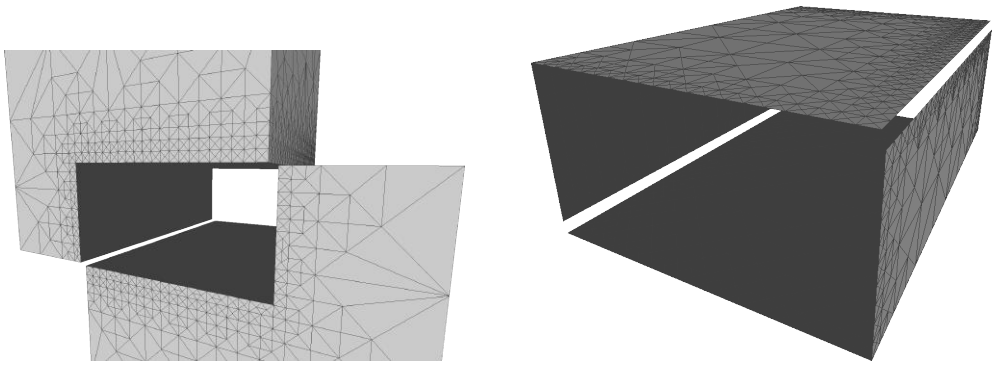
Test 5

V teste koplanárnych polygónov sa prejavil očakávaný problém s presnosťou aritmetiky. Preto bol využitý test potenciálneho prieniku, ktorý spoľahlivo odstráni kolidujúce trojuholníky. Po odstránení oblasti prieniku počas operácií zjednotenie a prienik však vzniknú

medzi hranicami veľké medzery (obrázok 8.11). Z pohľadu implementovaného algoritmu nie je takáto situácia ideálna, avšak algoritmus zaceľovania dier spoľahlivo hranice spojil. Problematická je operácia prienik, kde odstránením ostrých hrán vznikne nerovnomerný okraj výstupného modelu. Túto situáciu by bolo možné riešiť dodatočným spracovaním (*post-processing*), ktorý by zacelenú oblasť znova zjemnil a premietol na pôvodné modely.



Obr. 8.10: Výsledky testu 5: zjednotenie, rozdiel, prienik (zľava doprava).



(a) Medzera medzi hranicami pri operácií zjednotenie.

(b) Medzera medzi hranicami pri operácií prienik.

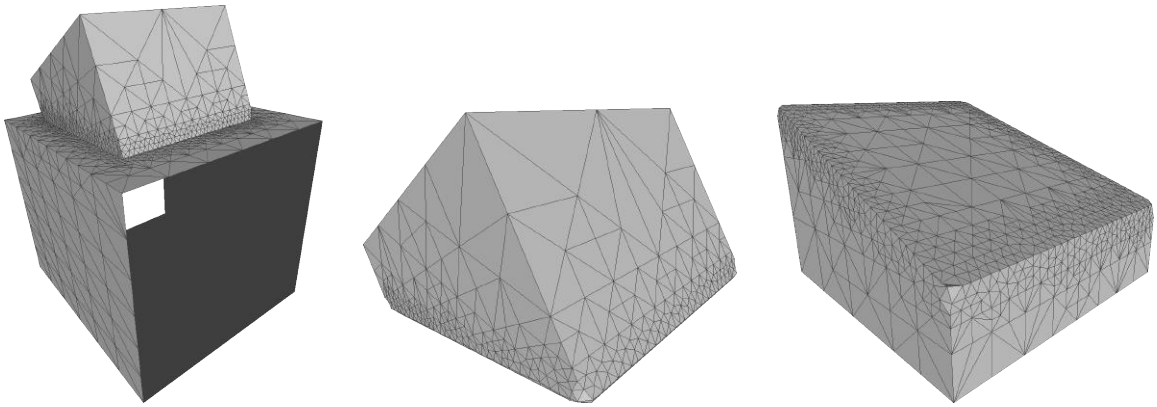
Obr. 8.11: Veľké medzery medzi hranicami v teste 5.

Zvolené parametre :

- Presnosť: 0.25
- Počet iterácií zjemňovania: 6
- Test priesečníku trojuholníkov: Potenciálny prienik

Test 6

Test číslo 6 prebehol bez problémov. Výstupný model operácie zjednotenie je otvorený, čo je však očakávaný výstup. Tento príklad demonštruje schopnosť algoritmu pracovať s otvorenými vstupnými modelmi.



Obr. 8.12: Výsledky testu 6: zjednotenie, rozdiel, prienik (zľava doprava).

Zvolené parametre :

- Presnosť: 0.1
- Počet iterácií zjemňovania: 15

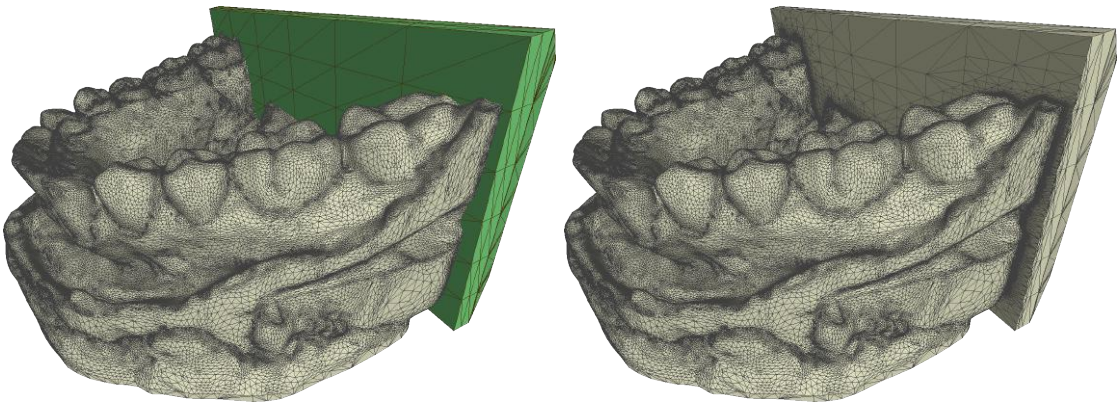
Test 7

Test takmer zhodných modelov hodnotím ako neúspešný, aj keď je výsledkom operácie rozdiel očakávaný výstup. Krok odstránenie oblasti prieniku v každej operácii odstráni celé vstupné modely. To však nie je korektné chovanie, keďže výstupom operácií zjednotenie a prenik sú potom prázdne modely. Ošetrenie podobných vstupných situácií môže byť námetom na ďalší vývoj knižnice.

8.3 Rozšírené testovanie

V spolupráci s firmou 3Dim Laboratory bola pripravená dátová sada určená na pokročilejšie testovanie. Jednotlivé testy sa snažia modelovať praktické situácie, s ktorými firma pracuje. Detailnú vizualizáciu môže čitateľ nájsť v prílohe B. Dátová sada obsahuje celkom tri modelové situácie:

- Pridanie podpory pre 3D tlač k modelu čeľusti (obrázok 8.13)
- Vloženie implantátu do ďasna pacienta (obrázok 8.14)
- Pridanie zámkov strojčeku na zuby (obrázok 8.15)

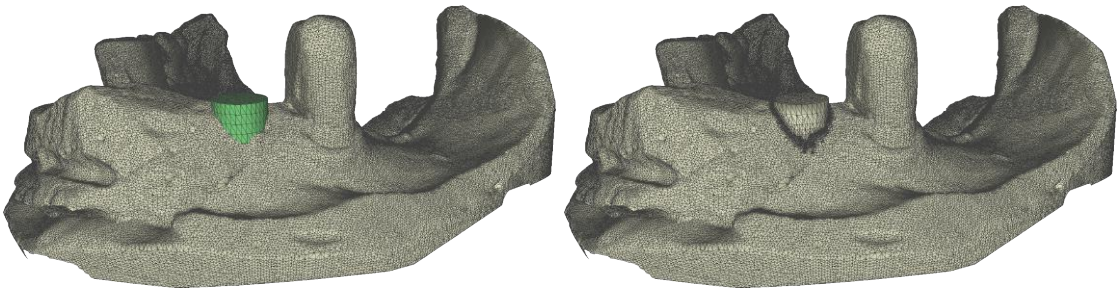


(a) Vstup testu. (b) Výstup testu.

Obr. 8.13: Test pridanie podpory pre 3D tlač k modelu čeľusti.

Prvou časťou dátovej sady sú autentické modely čeľusti pacientov, ktoré poskytla firma 3Dim Laboratory. Druhou časťou sú modely implantátov a podpora pre 3D tlač. Z licenčných dôvodov nemohli byt použité reálne modely implantátov používané v praxi. Tie boli nahradené hrubými aproximáciami ich tvaru.

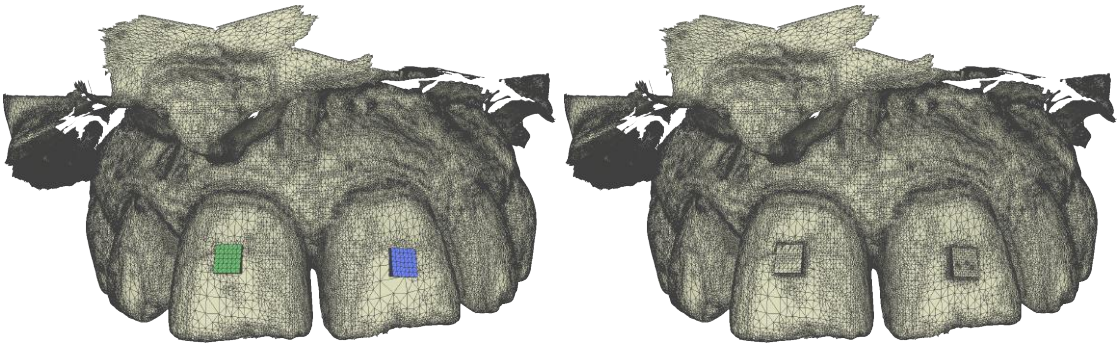
Vo všetkých testových situáciách bola použitá operácia zjednotenie. Testy prebehli bezproblémovo. Jediným nastaveným parametrom operácií bola presnosť, ktorá bola vždy odhadnutá na základe povahy vstupných modelov. Tretí test demonštruje schopnosť knižnice pracovať s viacerými vstupnými modelmi naraz.



(a) Vstup testu. (b) Výstup testu.

Obr. 8.14: Test vloženia implantátu do ďasna pacienta.

Cieľom týchto testov nebolo ďalšie meranie časovej a pamäťovej náročnosti. Účelom tohto testovania bolo ukázať, že knižnica môže byť využitá v praxi. Vstupné modely čeľusti nie sú ani z ďaleka ideálnymi uzavretými modelmi. V dvoch prípadoch sa jedna iba o povrchové skeny obsahujúce diery a *non-manifold* trojuholníky. Implementovaná knižnica napriek tomu dokázala vo všetkých prípadoch previesť požadované operácie s korektným výstupom.



(a) Vstup testu. (b) Výstup testu.

Obr. 8.15: Test pridanie zámkov strojčeku na zuby.

Kapitola 9

Záver

Cieľom tejto práce bolo vytvoriť knižnicu adaptívnych booleovských operácií. Booleovské operácie nad 3D polygonálnymi sieťami sa dajú považovať za nie uspokojivo vyriešený problém počítačovej geometrie. Využívajú sa v celej rade aplikácií, od jednoduchých 3D modelovacích nástrojov, cez zložité CAD systémy, až po rôzne medicínske aplikácie.

Existuje veľa rôznych prístupov k výpočtu booleovských operácií a dostupných riešení. Každá metóda a riešenie ma svoje silné a slabé stránky. V rámci prieskumu súčasného stavu boli otestované dve knižnice a dva modelovacie nástroje implementujúce booleovské operácie. Ani v jednom prípade sa však nedá hovoriť o 100%-nej úspešnosti.

V tejto práci bolo navrhnuté a implementované riešenie v podobe *C++* knižnice, ktoré je dostatočne robustné voči aritmetickým nepresnostiam, podporuje prácu s otvorenými modelmi a dokáže pracovať s viacerými vstupmi naraz. Výsledná knižnica bolo podrobená rovnakému testovaniu ako spomínané existujúce riešenia. Výsledky testovania sa dajú považovať za konkurencie schopné. Implementovaná knižnica bola navyše testovaná na rozšírenej dátovej sade, ktorá modeluje praktické prípady použitia vo firme 3Dim Laboratory. Výsledné riešenie bolo úspešné vo viac ako 90% testovaných prípadov.

Ďalší vývoj sa bude sústrediť na vylepšenie implementácie *remeshing* operácií, ktoré sú jedným zo stavebných kameňov knižnice. Významné urýchlenie priniesla paralelizácia výpočtu, ktorá však môže byť ďalej rozvinutá. Nezanedbateľným je aj vylepšenie zachovania ostrých detailov a spresnenie výsledkov.

Literatúra

[1] Autodesk Meshmixer. https://www.meshmixer.com/, [cit. 2018-11-01].

[2] Autodesk Meshmixer Boolean Operations. http://www.mmmanual.com/boolean/, [cit. 2018-11-01].

[3] Carve Library. https://code.google.com/archive/p/carve/, [cit. 2018-11-01].

[4] The CGAL Project. https://www.cgal.org/, [cit. 2018-11-01].

[5] Cork Boolean Library. https://github.com/gilbo/cork, [cit. 2018-11-01].

[6] The GNU Multiple Precision Arithmetic Library. https://gmplib.org/, [cit. 2018-11-01].

[7] Netfabb - Additive manufacturing and design software. https://www.autodesk.com/products/netfabb/overview, [cit. 2018-11-01].

[8] OpenMesh. https://www.openmesh.org/, [cit. 2018-11-01].

[9] VTK - The Visualization Toolkit. https://www.vtk.org/, [cit. 2018-11-01].

[10] Adams, B.; Dutré, P. Interactive boolean operations on surfel-bounded solids. 2003, 22, 3.

[11] Botsch, M.; Kobbelt, L. A remeshing approach to multiresolution modeling. 2004.

[12] Campen, M.; Kobbelt, L. Exact and Robust (Self-) Intersections for Polygonal Meshes. 2010, 29, 2.

[13] Devillers, O.; Guigue, P. : *Faster triangle-triangle intersection tests*. Dizertační práce, 2002.

[14] Eberly, D. : Distance between point and triangle in 3D. *Magic Software, http://www. magic-software. com/Documentation/pt3tri3. pdf*, 1999.

[15] Fang, S.; Zhu, X.; Bruderlin, B. : Robustness in solid modeling-a tolerance based. Technická zpráva, intuitionistic approach. Tech. Rep. UUCS 92-030 (submitted), Computer Science Department, University of Utah, 1992.

[16] Hachenberger, P.; Kettner, L.; Mehlhorn, K. : Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. *Computational Geometry*, roč. 38, č. 1-2, 2007: s 64–99.

[17] Hoppe, H. H. : Mesh simplification and construction of progressive meshes. Červenec 27 1999, uS Patent 5,929,860.

[18] Hu, C.-Y.; Patrikalakis, N. M.; Ye, X. : Robust interval solid modelling part I: representations. *Computer-Aided Design*, roč. 28, č. 10, 1996: s 807–817.

[19] Hu, C.-Y.; Patrikalakis, N. M.; Ye, X. : Robust interval solid modelling part II: boundary evaluation. *Computer-Aided Design*, roč. 28, č. 10, 1996: s 819–830.

[20] Jacobson, A.; Kavan, L.; Sorkine-Hornung, O. : Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)*, roč. 32, č. 4, 2013: str. 33.

[21] Ju, T.; Udeshi, T. Intersection-free contouring on an octree grid. 2006, 3.

[22] Kobbelt, L. P.; Botsch, M.; Schwanecke, U.; aj. Feature sensitive surface extraction from volume data. 2001.

[23] Lo, S.; Wang, W. : Finite element mesh generation over intersecting curved surfaces by tracing of neighbours. *Finite elements in analysis and design*, roč. 41, č. 4, 2005: s 351–370.

[24] Mäntylä, M. : Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics (TOG)*, roč. 5, č. 1, 1986: s 1–29.

[25] Mei, G.; Tipper, J. C. : Simple and robust boolean operations for triangulated surfaces. arXiv preprint. *arXiv*, roč. 1308, 2013.

[26] Möller, T. : A fast triangle-triangle intersection test. *Journal of graphics tools*, roč. 2, č. 2, 1997: s 25–30.

[27] Pavić, D.; Campen, M.; Kobbelt, L. Hybrid booleans. 2010, 29, 1.

[28] Requicha, A. A.; Voelcker, H. B. : Solid modeling: a historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, , č. 2, 1982: s 9–24.

[29] Requicha, A. A.; Voelcker, H. B. : Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, roč. 73, č. 1, 1985: s 30–44.

[30] Schaefer, S.; Ju, T.; Warren, J. : Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics*, roč. 13, č. 3, 2007: s 610–619.

[31] Schifko, M.; Jüttler, B.; Kornberger, B. Industrial application of exact boolean operations for meshes. 2010.

[32] Schmidt, R.; Brochu, T. : Adaptive mesh booleans. *arXiv preprint arXiv:1605.01760*, 2016.

[33] Segal, M. : Using tolerances to guarantee valid polyhedral modeling results. *ACM SIGGRAPH Computer Graphics*, roč. 24, č. 4, 1990: s 105–114.

[34] Shade, J.; Gortler, S.; He, L.-w.; aj. Layered depth images. 1998.

[35] Tayebi, A.; Perez, J. G.; Diego, I. G.; aj. Boolean operations implementation over 3D parametric surfaces to be included in the geometrical module of an electromagnetic solver. 2011.

[36] Tropp, O.; Tal, A.; Shimshoni, I. : A fast triangle to triangle intersection test for collision detection. *Computer Animation and Virtual Worlds*, roč. 17, č. 5, 2006: s 527–535.

[37] Varadhan, G.; Krishnan, S.; Kim, Y. J.; aj. Feature-sensitive subdivision and isosurface reconstruction. 2003.

[38] Varadhan, G.; Krishnan, S.; Sriram, T.; aj. Topology preserving surface extraction using adaptive subdivision. 2004.

[39] Wang, C. C. : Approximate boolean operations on large polyhedral solids with partial mesh reconstruction. *IEEE Transactions on visualization and computer graphics*, roč. 17, č. 6, 2011: s 836–849.

[40] Yang, P.; Qian, X. : Direct boolean intersection between acquired and designed geometry. *Computer-Aided Design*, roč. 41, č. 2, 2009: s 81–94.

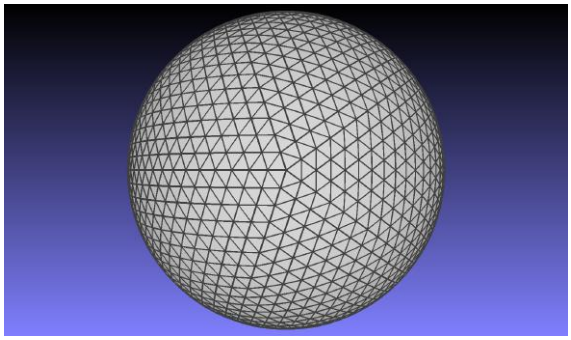
[41] Zhang, N.; Hong, W.; Kaufman, A. Dual contouring with topology-preserving simplification using enhanced cell representation. 2004.

[42] Zhou, L.; Wang, D.; Sheng, Y.; aj. Three dimensional Boolean operation based on L-Rep model. 2010.

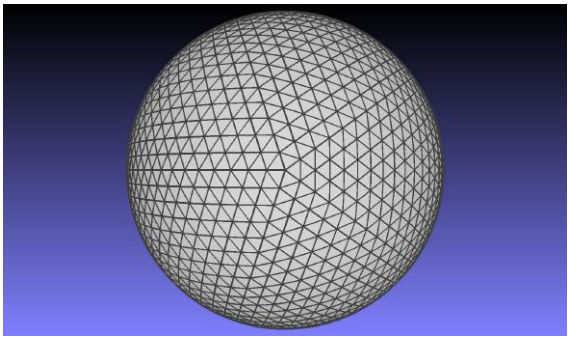
Príloha A

Dátová sada na testovanie

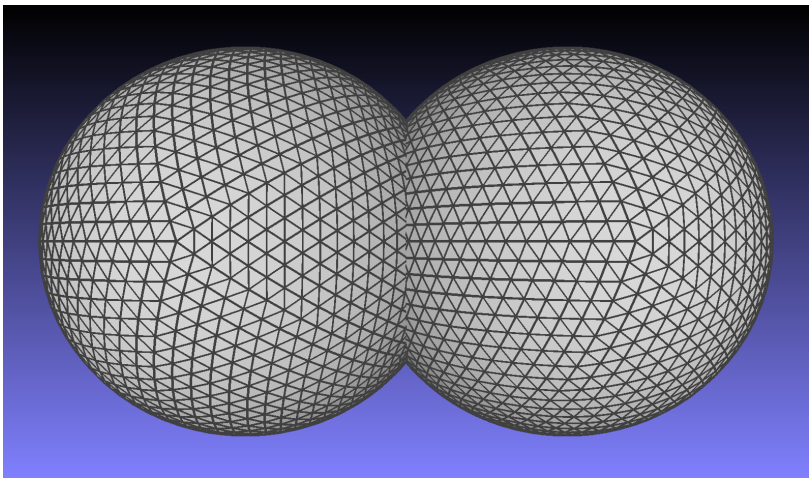
Test 1 - Pretínajúce sa sféry



Obr. A.1: Model A1
Počet trojuholníkov: 5,120

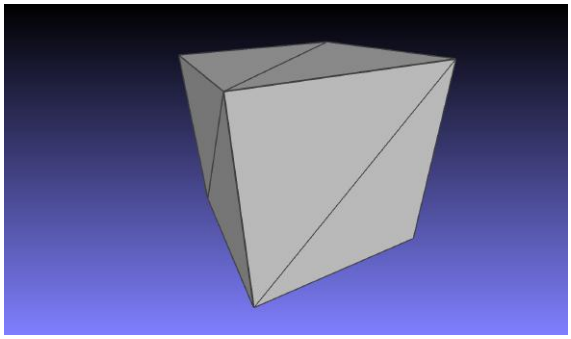


Obr. A.2: Model B1
Počet trojuholníkov: 5,120

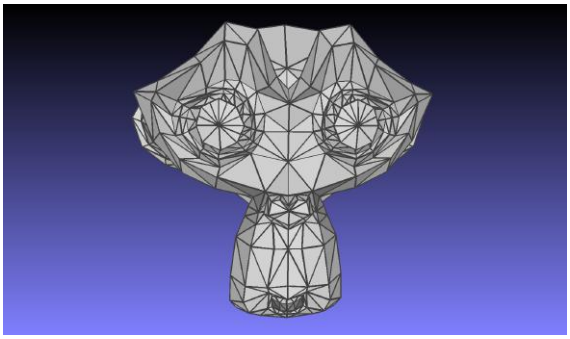


Obr. A.3: Test 1

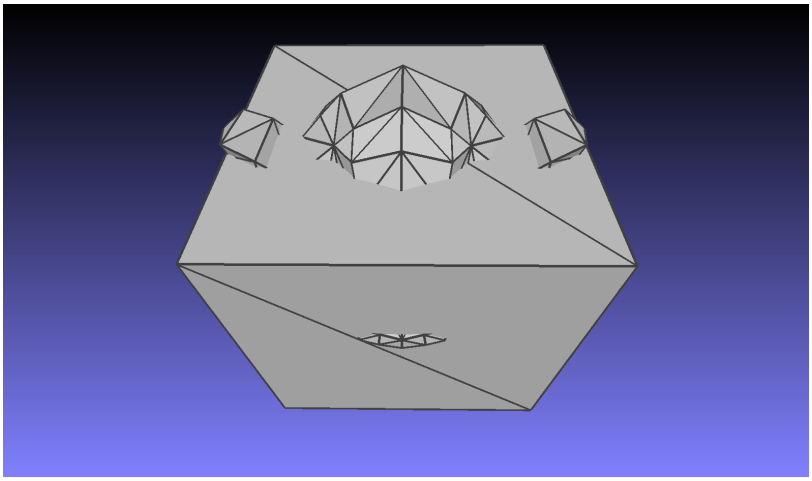
Test 2 - Netriviálny prienik



Obr. A.4: Model A2
Počet trojuholníkov: 12

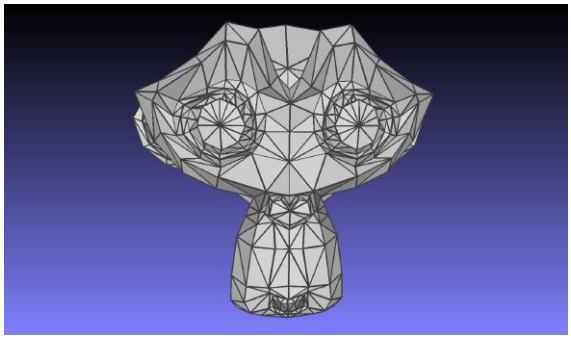


Obr. A.5: Model B2
Počet trojuholníkov: 1,026

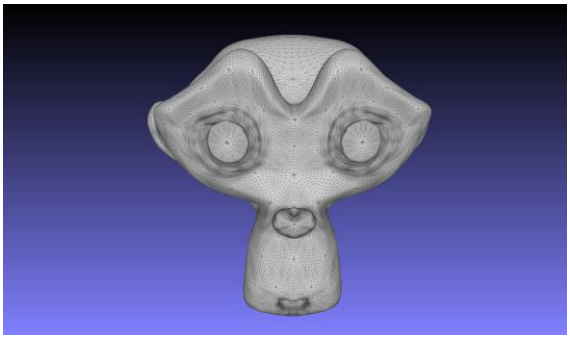


Obr. A.6: Test 2

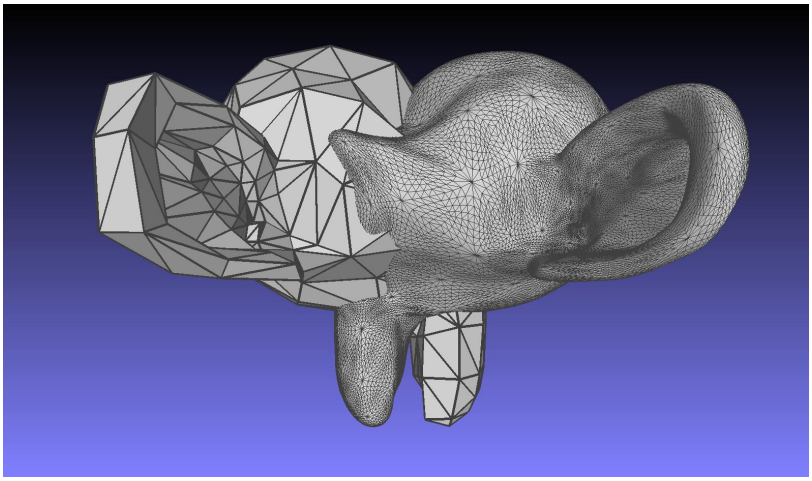
Test 3 - Jednoduchý a zložitý model



Obr. A.7: Model A3
Počet trojuholníkov: 1,026

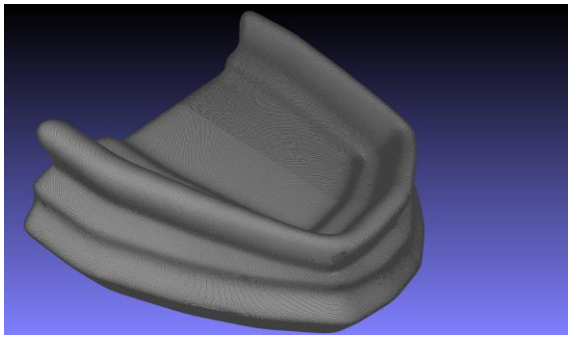


Obr. A.8: Model B3
Počet trojuholníkov: 98,522

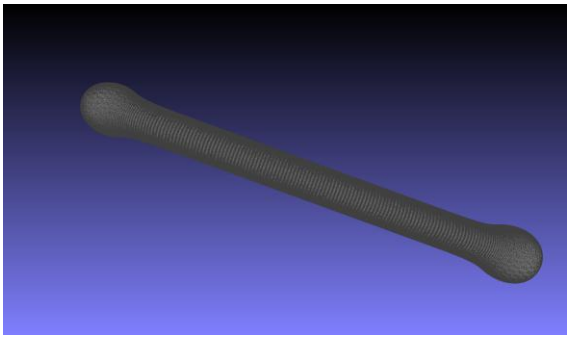


Obr. A.9: Test 3

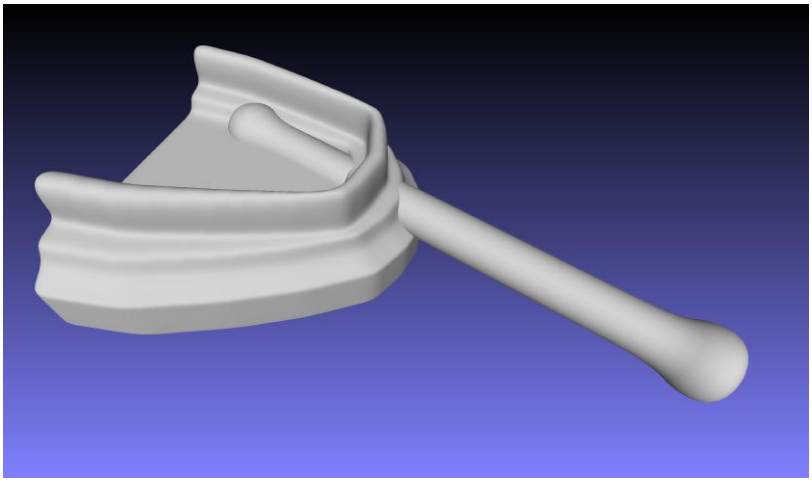
Test 4 - Modely s veľkým počtom trojuholníkov



Obr. A.10: Model A4
Počet trojuholníkov: 1,568,768

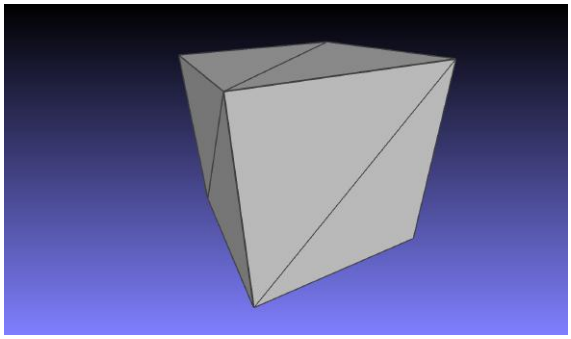


Obr. A.11: Model B4
Počet trojuholníkov: 270,336

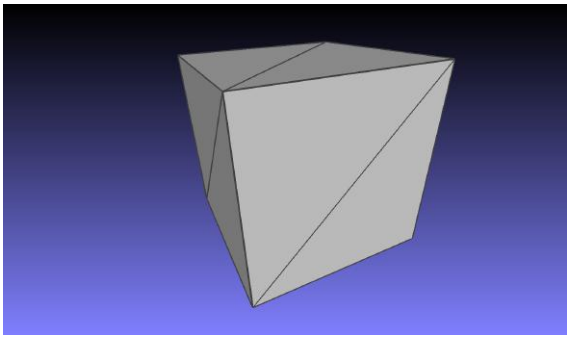


Obr. A.12: Test 4

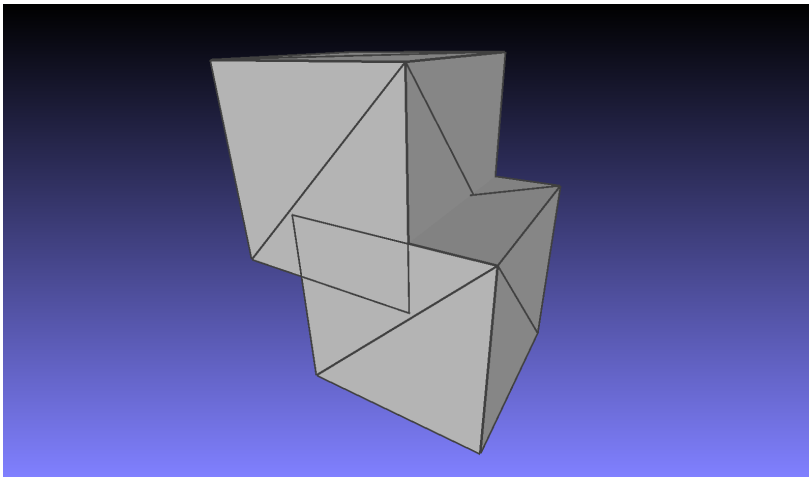
Test 5 - Koplanárne polygóny



Obr. A.13: Model A5
Počet trojuholníkov: 12

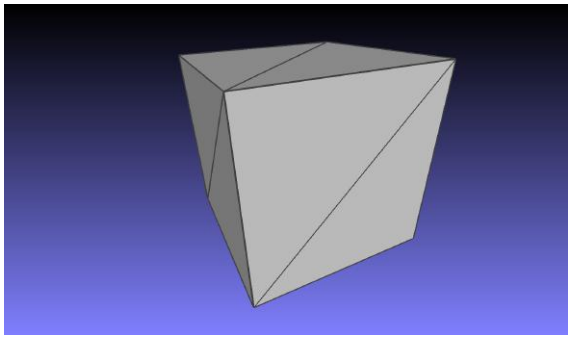


Obr. A.14: Model B5
Počet trojuholníkov: 12

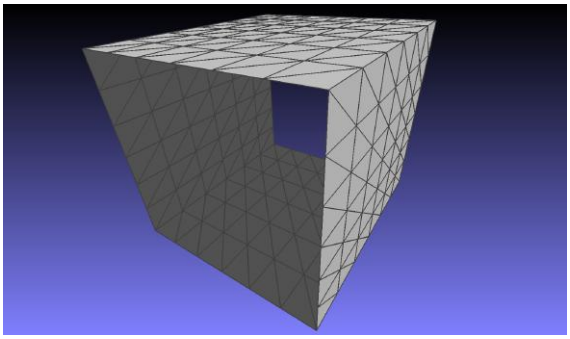


Obr. A.15: Test 5

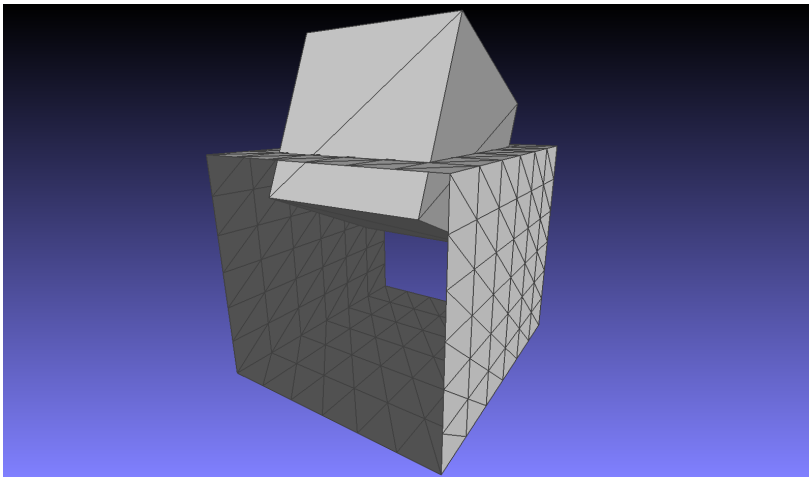
Test 6 - Otvorený a uzavrený model



Obr. A.16: Model A6
Počet trojuholníkov: 12

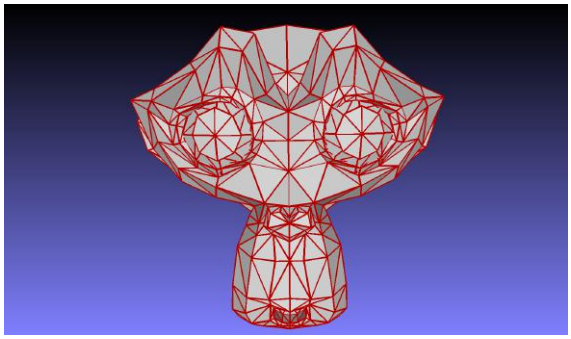


Obr. A.17: Model B6
Počet trojuholníkov: 288

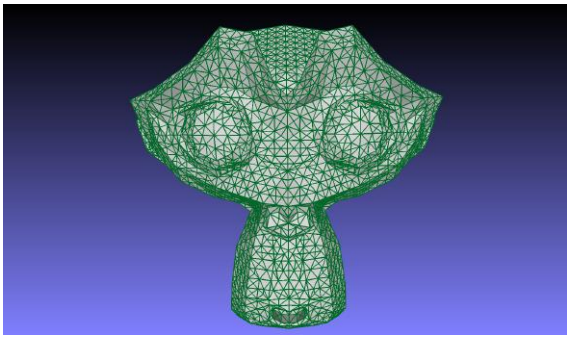


Obr. A.18: Test 6

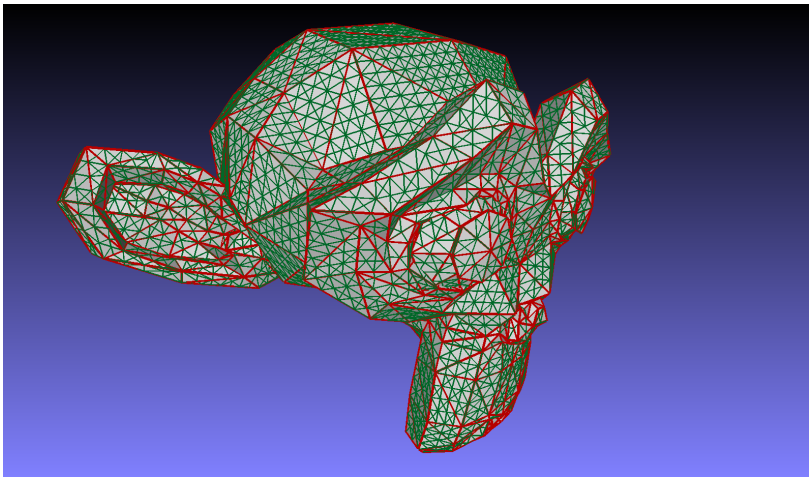
Test 7 - Dva rovnaké modely s rozdielnou úrovňou teselácie, navyše obsahujúce diery



Obr. A.19: Model A7
Počet trojuholníkov: 968



Obr. A.20: Model B7
Počet trojuholníkov: 13,866

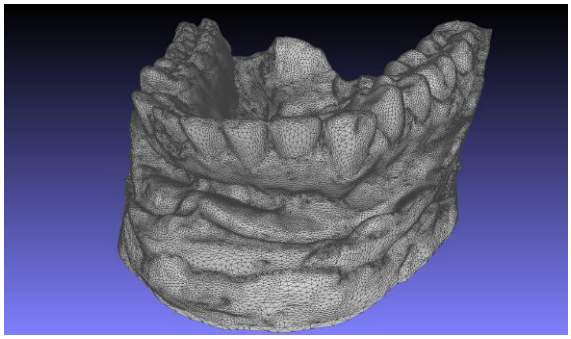


Obr. A.21: Test 7

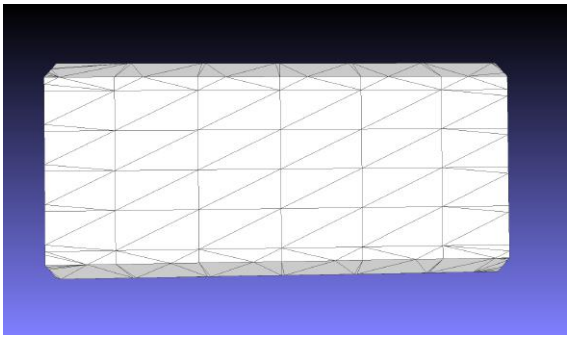
Príloha B

Rozšírená dátová sada na testovanie

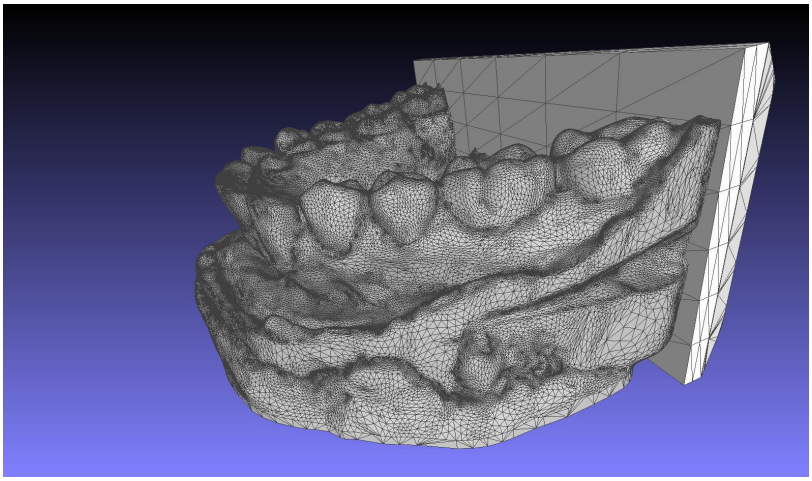
Test 1 - Pridanie podpory pre 3D tlač k modelu čeľusti



Obr. B.1: Model A1
Počet trojuholníkov: 200,314

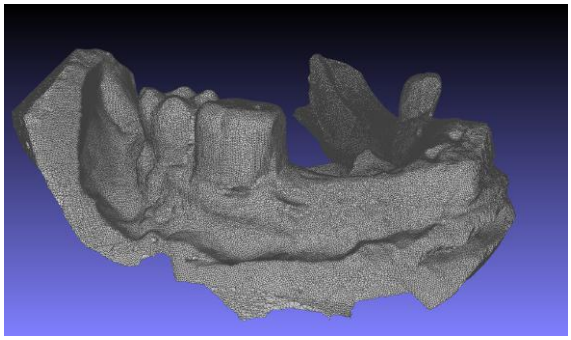


Obr. B.2: Model B1
Počet trojuholníkov: 510

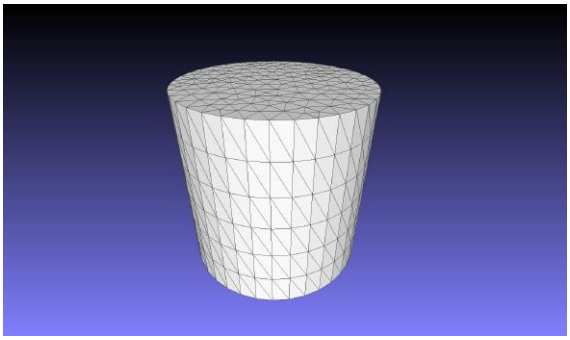


Obr. B.3: Test 1

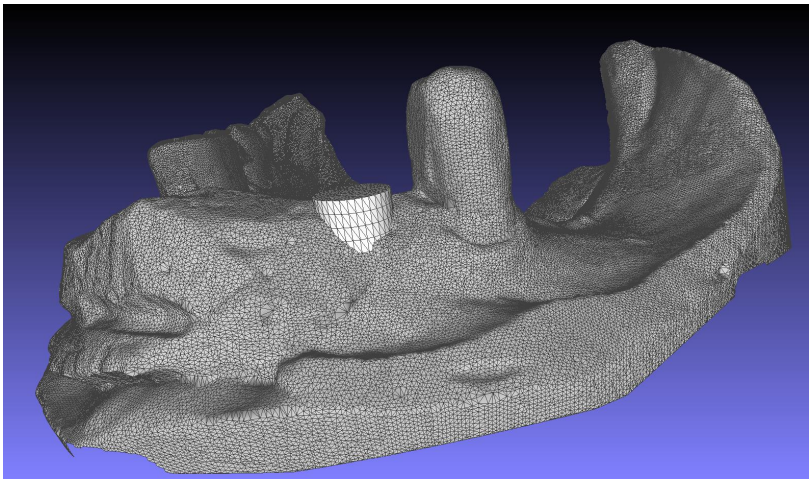
Test 2 - Vloženie implantátu do ďasna pacienta



Obr. B.4: Model A2
Počet trojuholníkov: 239,621

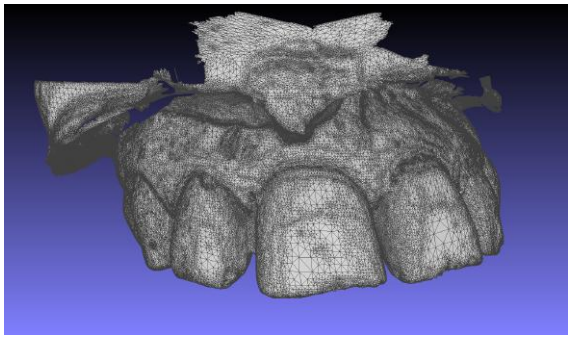


Obr. B.5: Model B2
Počet trojuholníkov: 940

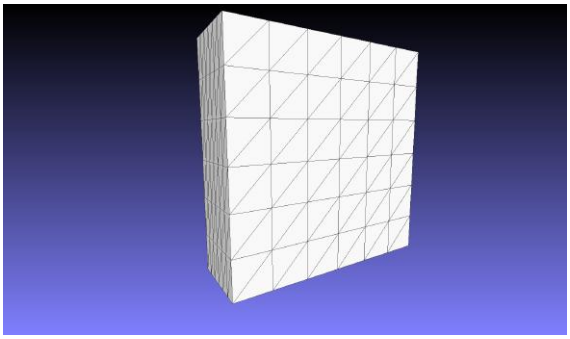


Obr. B.6: Test 2

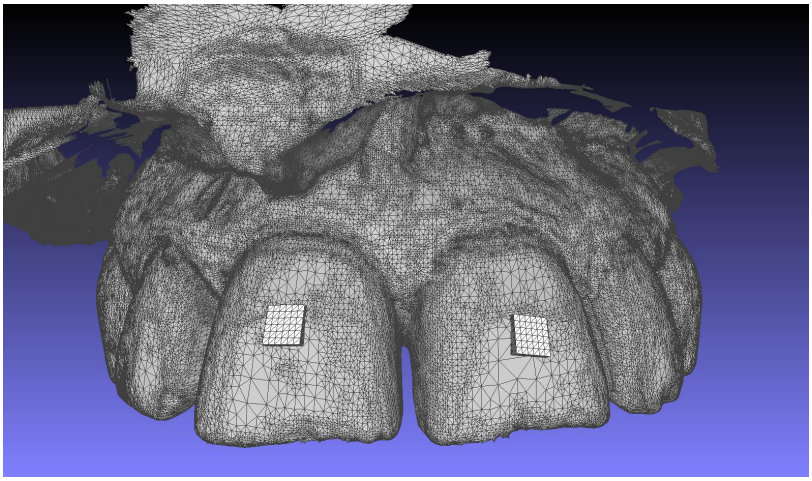
Test 3 - Pridanie zámkov strojčeku na zuby



Obr. B.7: Model A3
Počet trojuholníkov: 1,044,250



Obr. B.8: Model 31
Počet trojuholníkov: 432



Obr. B.9: Test 3

Príloha C

Výsledky testovania existujúcich riešení

CGAL

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	7,8	308	áno	
2	áno	0,471	45	áno	
3	áno	49,3	2700	áno	3 self-intersections
4	nie	N/A	6500	N/A	pád OS
5	áno	0,063	16	áno	
6	nie	N/A	N/A	N/A	
7	áno	63	1700	nie	741 self-intersections

Tabuľka C.1: Výsledky testovania operácie **zjednotenie**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	5.3	281	áno	
2	áno	0.615	47	áno	
3	áno	51	2700	áno	8 self-intersections
4	nie	N/A	6500	N/A	pád OS
5	áno	0.06	15	áno	
6	nie	N/A	N/A	N/A	
7	áno	57	1700	áno	

Tabuľka C.2: Výsledky testovania operácie **rozdiel**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	4.8	280	áno	
2	áno	0.57	48	áno	
3	áno	50	2700	áno	8 self-intersections
4	nie	N/A	6500	N/A	pád OS
5	áno	0.07	12	áno	
6	nie	N/A	N/A	N/A	
7	áno	64	1700	nie	6952 self-intersections

Tabuľka C.3: Výsledky testovania operácie **prienik**

VTK

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	0.63	10	áno	
2	áno	0.163	6	nie	model obsahuje diery
3	áno	2.57	63	áno	88 self-intersections
4	áno	56.56	820	áno	89 self-intersections
5	nie	N/A	N/A	N/A	pád programu
6	áno	0.063	6	áno	
7	nie	N/A	N/A	N/A	pád programu

Tabuľka C.4: Výsledky testovania operácie **zjednotenie**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	0.63	9,5	áno	
2	áno	0.17	6	nie	model obsahuje diery
3	áno	2.63	64	áno	39 self-intersections
4	áno	50.58	820	áno	123 self-intersections
5	nie	N/A	N/A	N/A	pád programu
6	áno	0.099	6	áno	
7	nie	N/A	N/A	N/A	pád programu

Tabuľka C.5: Výsledky testovania operácie **rozdiel**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	0.622	10	áno	
2	áno	0.184	6	nie	model obsahuje diery
3	áno	2.62	66	áno	54 self-intersections
4	áno	23,847	820	áno	131 self-intersections
5	nie	N/A	N/A	N/A	pád programu
6	áno	0.065	6	áno	
7	nie	N/A	N/A	N/A	pád programu

Tabuľka C.6: Výsledky testovania operácie **prienik**

Netfabb

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	<1	N/A	áno	
2	áno	<1	N/A	áno	
3	áno	2	N/A	áno	8 self-intersections
4	áno	149	∼ 4200	áno	64 self-intersections, nárast využitej pamäte o 4.2 GB
5	áno	<1	N/A	áno	
6	nie	N/A	N/A	N/A	
7	áno	20	N/A	nie	model obsahuje diery

Tabuľka C.7: Výsledky testovania operácie **zjednotenie**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	<1	N/A	áno	
2	áno	<1	N/A	áno	
3	áno	2	N/A	áno	20 self-intersections
4	áno	153	∼ 4200	áno	43 self-intersections, nárast využitej pamäte o 4.2 GB
5	áno	<1	N/A	áno	
6	nie	N/A	N/A	N/A	
7	áno	20	N/A	áno	

Tabuľka C.8: Výsledky testovania operácie **rozdiel**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	<1	N/A	áno	
2	áno	<1	N/A	áno	
3	áno	2	N/A	áno	20 self-intersections
4	áno	153	∼ 4200	áno	43 self-intersections, nárast využitej pamäte o 4.2 GB
5	áno	<1	N/A	áno	
6	nie	N/A	N/A	N/A	
7	áno	20	N/A	nie	10 non-manifold hrán, 2096 self-intersections

Tabuľka C.9: Výsledky testovania operácie **prienik**

Meshmixer

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	N/A	N/A	áno	
2	áno	N/A	N/A	nie	609 non-manifold hrán
3	áno	N/A	N/A	nie	10 non-manifold hrán, model obsahuje diery
4	áno	N/A	N/A	áno	42 self-intersections
5	áno	N/A	N/A	nie	
6	áno	N/A	N/A	áno	
7	nie	N/A	N/A	N/A	

Tabuľka C.10: Výsledky testovania operácie **zjednotenie**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	N/A	N/A	áno	
2	áno	N/A	N/A	nie	3661 non-manifold hrán
3	áno	N/A	N/A	nie	7441 self-intersections, 4 non-manifold hrany
4	áno	N/A	N/A	áno	42 self-intersections
5	áno	N/A	N/A	nie	
6	áno	N/A	N/A	áno	
7	nie	N/A	N/A	N/A	

Tabuľka C.11: Výsledky testovania operácie **rozdiel**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	N/A	N/A	áno	
2	áno	N/A	N/A	nie	836 non-manifold hrán
3	áno	N/A	N/A	nie	7 non-manifold hrán
4	áno	N/A	N/A	áno	4 self-intersections
5	áno	N/A	N/A	nie	
6	áno	N/A	N/A	áno	
7	nie	N/A	N/A	N/A	

Tabuľka C.12: Výsledky testovania operácie **prienik**

Vlastná implementácia

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	1.95	10.3	áno	37 self-intersections
2	áno	6.71	30.8	áno	180 self-intersections
3	áno	152.1	310.6	áno	2120 self-intersections
4	áno	27.16	438.5	áno	204 self-intersections
5	áno	1.47	7.8	áno	12 self-intersections
6	áno	0.72	2.6	áno	14 self-intersections
7	áno	1.35	20.8	nie	

Tabuľka C.13: Výsledky testovania operácie **zjednotenie**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	2.11	10.6	áno	35 self-intersections
2	áno	7.7	31.4	áno	322 self-intersections
3	áno	15.58	62.6	áno	2534 self-intersections
4	áno	73.7	441.1	áno	316 self-intersections
5	áno	0.46	6.1	áno	6 self-intersections
6	áno	0.59	2.7	áno	14 self-intersections
7	áno	1.35	20.8	áno	

Tabuľka C.14: Výsledky testovania operácie **rozdiel**

Č. testu	Úspešnoť operácie	Trvanie výpočtu [s]	Max. spotrebovaná pamäť [MB]	Korektný výstup	Poznámka
1	áno	1.84	10.4	áno	24 self-intersections
2	áno	12.48	57.7	áno	173 self-intersections
3	áno	111.74	257.6	áno	88 self-intersections
4	áno	42.32	440.7	áno	282 self-intersections
5	áno	1.58	8	áno	2 self-intersections
6	áno	1.27	2.7	áno	18 self-intersections
7	áno	1.35	20.8	nie	

Tabuľka C.15: Výsledky testovania operácie **prienik**

Príloha D

Obsah priloženého pamäťového média

Bin	Zložka so spustiteľnými binárnymi súbormi
Text	Zložka so zdrojovými súbormy technickej správy
Sources	Zložka so zdrojovými kódmi implementácie
Video	Zložka s videom
xcizma04_DIP.pdf	Text technickej správy